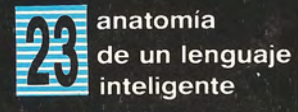
BIBLIOTECA BASICA INFORMATICA

FORTH



INGELEK

BIBLIOTECA BASICA INFORMATICA

FORTH

23 anatomía de un lenguaje inteligente

INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Monsalve.

Colaboradores:
Angel Segado
Casimiro Zaragoza
Fernando Ruíz
Francisco Ruíz
Jesús Pedraza
Juanjo Alba Ríos
Margarita Caffaratto
María Angeles Gálvez
Marina Caffaratto
Masé González Balandín
Patricia Mordini

Diseño:
Bravo/Lofish.

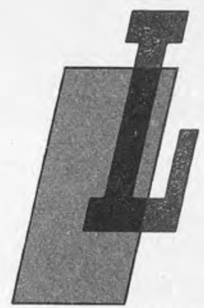
Dibujos:
José Ochoa.

ISBN del tomo: 84-85831-61-6
ISBN de la obra: 84-85831-31-4
Fotocomposición: Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-11.993-1986
Precio en Canarias, Ceuta y Melilla: 380 pts.



INDICE

PROLOGO



La primera pregunta que podríamos plantearnos es: ¿por qué otro lenguaje más? Cierto es que el BASIC, queramos o no, se está casi convirtiendo en el esperanto de los aficionados a la informática (a pesar de serlo de una manera bastante imperfecta si tenemos en cuenta la enorme cantidad de dialectos existentes); también es cierto que muchos lenguajes se olvidan, al tiempo que cada día aparecen otros diferentes. Paralelamente a todo esto van creándose partidarios a favor de uno u otro lenguaje. Pero no queremos sumarnos a la tónica general y contar las maravillas del FORTH por el mero hecho de haberle "tomado cariño". Nuestro propósito es explicar de una manera seria y objetiva el por qué del éxito de este lenguaje en aspectos donde los restantes han fallado.

Hay que hacer notar, en primer lugar, que el usuario muestra una clara tendencia a adoptar el lenguaje más sencillo y que le cree el menor número de problemas posibles. Fue ésta precisamente una de las razones que determinó el éxito del BASIC.

Lo primero que llama la atención del FORTH es la manera tan original e innovadora en que se trata el problema de la programación. Resulta, además, indispensable en ciertos tipos de aplicaciones. De estas dos cuestiones en particular trataremos en este libro. Ciertamente no es posible mostrar su potencialidad al completo ni podremos enseñarles cómo utilizar y aprovechar todas sus instrucciones; lo que queremos es dar una idea bastante clara acerca del estilo de programación FORTH y de lo que es posible o no hacer con él. Si alguna persona encuentra este tema interesante y desea profundizar más sobre ello podrá hallar una gran

variedad de libros en las librerías especializadas (vea también la bibliografía al final de este volumen).

Otra pregunta que puede surgir es: ¿para quién es útil el FORTH? La respuesta es bien sencilla: para quien odie los programas que abarcan diez páginas de listado, para aquellos que estén abiertos a nuevas concepciones, para los que deseen una ejecución más rápida de sus programas y para los que no aguantan la sintaxis tipo "sujeto-predicado-complemento".

Por último, tan sólo una recomendación. Para aprender bien un lenguaje, sea o no de ordenadores, son obviamente necesarios los manuales, pero también un mínimo de práctica. Hemos planteado el presente libro para que pueda ser leído sin tener que recurrir al uso de un ordenador para seguirlo. A pesar de ello, resulta obvio que llevar a la realidad, aunque sea de forma esporádica, lo explicado sobre cualquier ordenador hará menos penosa la etapa de aprendizaje y ofrecerá la posibilidad de intentar efectuar alguna variante. Supone una gran ventaja hoy en día el que para emprender la aventura del FORTH sea suficiente con un pequeño micro doméstico y poseer este económico, pero potente, software.

CAPITULO I

HISTORIA Y PRESENTE DEL FORTH

Un ordenador no es otra cosa que un conjunto de circuitos y cables que, para desarrollar sus funciones, necesita, además de ser conectado a la red, ser "revestido" por el software, es decir, las instrucciones. El ordenador entiende estas instrucciones como integrantes de un "idioma" concreto denominado código máquina, y las comprende y ejecuta de una manera tan rápida y eficiente que haría vacilar a cualquier mente humana. Naturalmente, si lo que pretendiéramos fuera que el ordenador comprendiera una orden dada en una lengua cualquiera utilizada por los humanos, el resultado sería el contrario.

Han sido así desarrollados los llamados lenguajes de programación, verdaderos idiomas dotados de gramática y sintaxis propias que si por un lado son fácilmente utilizables e inteligibles por los hombres, son por otro aptos para que el ordenador los traduzca en código máquina directamente interpretables por él. En BASIC, el utensilio interno que efectúa tal traducción se denomina intérprete. El intérprete toma de la memoria una instrucción, la transforma a código de máquina y la pasa al microprocesador; a continuación salta a la instrucción (o línea) siguiente y repite el proceso.

Naturalmente, todo esto requiere un tiempo que puede llegar a ser considerable en programas complejos e incluso prohibitivo cuando sea necesario efectuar operaciones y medidas inmediatas o, como suele decirse, en tiempo real. Este es muy a menudo el caso, por ejemplo, de máquinas herramientas, sistemas industriales o aparatos de medida controlados por microprocesador, que no toleran retrasos en la transmisión o recepción de datos.

Una solución podría ser la de los lenguajes compilados, como PASCAL o FORTRAN, pero en ellos las operaciones de búsqueda y corrección de errores son pesadas y complejas, y ocasionan molestias considerables al usuario.

El FORTH ofrece otra alternativa. Aunque las subrutinas que componen un programa FORTH son compiladas y así almacenadas, el trabajo con el lenguaje se lleva a cabo como si fuera interpretado. Al igual que en éste, en otros muchos aspectos del FORTH hay que olvidarse de ideas preconcebidas y abrirse a lo original.

El lenguaje FORTH fue creado entre los años 1965 y 1970 por Charles H. Moore y Elisabeth Rather en el National Radio Astronomy Observatory de Kitt Peak, Arizona. Tal y como Moore mismo cuenta, en aquel período a menudo tenía que trabajar en diferentes aplicaciones, como cálculo de trayectorias de cuerpos en órbita, cromatografías, análisis de espectros de emisión, cálculos de probabilidad, conservación de datos en archivo y análisis estadísticos y a veces financieros sobre los mismos. Se dio cuenta entonces que con los lenguajes que en aquel tiempo tenía a su disposición no habría podido producir en toda su vida más que unos tres o cuatro programas serios, desperdiciando inútilmente mucho tiempo.

Comprendió que necesitaba unos medios capaces de disminuir el tiempo y ritmo de programación. La idea, concretada en un principio en la realización de una serie de subrutinas de utilidad, dio lugar, en 1968, a una entidad (todavía se trataba de un bloque de subrutinas en lenguaje máquina, así que sería exagerado definirlo como lenguaje) que fue llamado FORTH.

El nombre deriva del hecho de que, en aquella época de ordenadores de la tercera generación, Moore quería considerar su creación (y no andaba muy equivocado) como perteneciente a la cuarta (FOURTH) generación. La primera edición del lenguaje fue implementada sobre un IBM 1130, que tan sólo permitía identificadores de una longitud máxima de cinco letras.

En 1971, mientras escribía un programa de adquisición de datos para un radiotelescopio de la empresa NRAO, Moore decidió añadir un compilador al sistema; dos años más tarde llegó a la multiprogramación. En el mismo período, Moore puso a punto un programa para la adquisición automática y continua de datos de un nuevo radiotelescopio instalado en Kitt Peak. A éste se debe el descubrimiento de al menos la mitad de los módulos interestelares (space dust) conocidos en la actualidad.

Empujados por una parte por su interés en astronomía y por otra por los no muy estimulantes sueldos de la NRAO, tanto Moore como Rather y otros compañeros dejaron en el mismo año el observatorio para fundar, en Hermosa Beach (California), la sociedad FORTH Inc. Esta se dedicó en sus comienzos a las aplicaciones astronómicas, pero pronto empezó a extenderse por nuevos sectores, llegando a abarcar hoy en día prácticamente todos los campos de la programación.

El hecho de que en los comienzos se interesaran en astronomía había determinado que Moore y su lenguaje fueran bastante conocidos en todo el mundo. En 1976 se constituyó EFUG (European FORTH Users Group). Ya desde su primer encuentro internacional decidieron formar el FORTH STANDARD TEAM (Grupo de FORTH estándar), con el objetivo de establecer un conjunto mínimo de comandos que fuera incluido en todos los sistemas y versiones.

En 1978 en San Carlos, California, un grupo de programadores fundaron el FIG (FORTH Interest Group), cuyo fin era promocionar el uso de este lenguaje por medio de seminarios, publicaciones e intercambio de ideas y experiencias a través de un periódico bimestral, el "FORTH Dimensions". El FIG pone además a disposición de sus asociados, y prácticamente al precio de coste, los listados fuente del lenguaje para diferentes microprocesadores y los proporciona, si así se desea, ya listos para su uso. En 1984 el FIG contaba ya en todo el mundo con más de 10.000 inscripciones oficiales.

El FORTH STANDARD TEAM, que aún pervive, se encarga de poner al día periódicamente el estándar del lenguaje (la versión más reciente es el FORTH-83). Por su parte el FIG se encarga, casi simultáneamente, de publicar un estándar, resumido en el Fig-Forth Installation Manual, escrito por William F. Ragdale. Afortunadamente, las características de este lenguaje son tales que las diferencias entre los diversos dialectos (como PoliForth, MSSForth, GF-Forth) son fácilmente integrables por los demás.

CAPITULO II

LA PALABRA (WORD), BASE DEL FORTH



Puede que les parezca increíble, pero en la prehistoria del lenguaje FORTH aparece Isaac Newton. Este científico genial, en un tiempo en que el cálculo simbólico estaba aún en sus albores (el uso de las expresiones literales fue introducido por el francés Viete en 1590), fue el primero en proponer el concepto de "Word" (palabra). Se trataba de una notación particular según la cual hacía corresponder a palabras preestablecidas (word, en inglés) operaciones, secuencias numéricas e incluso procedimientos matemáticos complicados.

La cosa terminó así, sobre todo porque resultaba más difícil recordar a qué correspondía cada palabra que describir la operación en sí. Los vocablos de Newton, que en su tiempo parecieron una complicación inútil, poseen la misma función que las palabras (words) en el FORTH. Es decir: hacen posible definir nombres, palabras o secuencias peculiares de caracteres a los que corresponden una determinada secuencia de operaciones.

Queremos precisar tan sólo una cosa antes de seguir: las palabras (words) figurarán en este libro siempre en mayúsculas. En caso de que surgieran complicaciones o se pudiera dar lugar a confusiones (cosa que puede ocurrir, por ejemplo, cuando una "word" de un solo carácter se nombra en un texto) las pondremos encerradas entre paréntesis que, naturalmente, no tendrán que ser tecleados en el momento de utilizarlas. De modo que aparecerán LEAVE, IF, BEGIN, pero también [;], [@] y [EXIT] en el caso de que este último pueda crear malentendidos dentro del texto.

Además, como ya dijimos, el lenguaje FORTH es, probablemente, el que se halla más desvinculado del ordenador sobre el

que está implementado; a pesar de ello, todo lo que propondremos (programas, words, subrutinas) ha sido probado y verificado en un ordenador Hewlett-Packard modelo 87XM y en un ITT 3710, además de un Spectrum 48K con un programa de Abersoft.

Cada lenguaje posee unos términos peculiares para describir qué órdenes hay que enviar desde el teclado a fin de que se ejecute una acción determinada. Las instrucciones del Assembler, las del Pascal o del BASIC, las listas o expresiones del LISP equivalen a lo que se llama, en FORTH, palabra (word).

La palabra es una secuencia de uno o varios caracteres alfanuméricos que determina siempre que aparece la misma secuencia de acciones, operaciones, etc. Por ejemplo, la palabra CR realiza el salto al principio de una línea (en impresora o en pantalla), la palabra [–] efectúa una sustracción entre dos números y pone el resultado en memoria, EMIT permite imprimir caracteres, etc.

Igual que cualquier otro lenguaje, todo sistema FORTH sea cual sea el dialecto a que pertenezca, posee un conjunto preestablecido de palabras, formando el llamado diccionario.

Los diccionarios contienen varios conjuntos (sets) de palabras a partir del "required word set", que son las palabras necesarias en cualquier implementación. En todo caso, en FORTH el diccionario no se limita a las palabras presentes en el sistema tal y como se ha comprado. El FORTH, por su naturaleza, permite ampliar su vocabulario con palabras de construcción original, escritas por el usuario, destinadas a cumplir funciones específicas o a ejecutar determinados procedimientos. Esto quiere decir que es posible, por ejemplo, crear palabras distintas de las originales uniendo algunas de las que ya existen en el vocabulario de base. A su vez estas nuevas creaciones se podrán unir para formar otras, consiguiendo así que efectúen procedimientos cada vez más complejos.

Todo esto lo había intuido Newton, pero su objetivo había sido crear códigos mnemotécnicos sintéticos en lugar de secuencias operativas. Trataba de buscar las fórmulas (que sólo aparecerán en el siglo XVIII) como instrumento de resumen y representación. El FORTH resume pero, sobre todo, lleva a cabo, mediante las palabras, operaciones, medidas, controles..., es decir, tanto acciones matemáticas como físicas.

Pongamos un ejemplo práctico: después de los primeros 1000 kilómetros, un coche nuevo es llevado a un mecánico para que le sean efectuados una serie de controles. Si el dueño, digámoslo así, se expresara en BASIC, diría al mecánico: quiero el cambio de aceite, el control de los frenos, el ajuste de las válvulas, la comprobación del nivel en la batería, etc. Es decir, señalaría una serie de operaciones, incluso las más complejas, llamándolas por su nombre.

Si el lenguaje utilizado fuera Assembler la comunicación resultaría de una complejidad exasperante, ya que tendría que describir minuciosamente cada operación: especificar que se ponga un contenedor debajo de la salida de aceite, desenroscar el tapón, dejar caer el aceite hasta la última gota, etc.

En FORTH, en cambio, basta con que se defina una sola palabra, por ejemplo llamada REVISION, estableciendo que a esta palabra correspondan toda una serie de revisiones y controles. Bastará con dirigirse al mecánico y pronunciar esta palabra para que todas las operaciones sean efectuadas y, lo que es más importante, la palabra REVISION permanecerá en el diccionario para cualquier necesidad futura, a menos que se quiera prescindir explícitamente de ella o destinarla a otros fines.

Esta estructura piramidal (o, como decimos en informática, "jerárquica") que permite la formación de palabras cada vez más complejas a partir de otras más sencillas, se podría comparar a las concavidades del terreno donde concluyan afluentes para formar cursos de agua cada vez más importantes. Tal estructura posee la ventaja de simplificar al máximo uno de los problemas de por sí más complejos en programación: la depuración de los errores sintácticos y lógicos (debug). Como el programa (o la palabra final; si piensa un poco verá que son la misma cosa) está, en su base, formado por palabras comprobadas una por una independientemente del contexto, la búsqueda de errores se facilita grandemente.

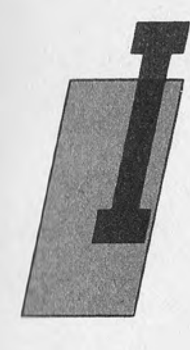
Siguiendo otra metáfora, más difundida, las palabras del FORTH son como perlas de un collar, ensartadas en un hilo y a su vez "cosidas" formando palabras cada vez más complejas. En efecto, la filosofía del FORTH es tal que algunos autores (vea la bibliografía) lo encuadran en la categoría de los llamados TIL (Threaded Interpretive Language), donde "threaded" significa, precisamente, cosido o enhebrado.

Una gran ventaja de este lenguaje es su modularidad y estructurabilidad; las palabras, una vez definidas, están siempre listas para su uso. Se parecen bastante a las "procedures" del PASCAL, pero esstán mucho más disponibles y utilizables.

Antes de ver de qué manera se construye una nueva palabra (la operación es de una sencillez tremenda: bastará tan sólo con definirla) será necesario volver a repasar algunos concepto y formular otros nuevos.

CAPITULO III

DENTRO DE LA PILA (STACK) CON LA NOTACION RPN



Imaginemos que queremos efectuar en una calculadora de bolsillo la operación

3 × 8

En la mayor parte de las calculadoras tendremos:

Secuencia	Display
3	3
×	3
8	8
=	24

Este tipo de inserción de los datos (números) y de los operadores (×,=) se llama notación (o sistema operativo) algebraico (SOA, o, en inglés, AOS). En ella se basan la mayor parte de las calculadoras que hay en el mercado y es la forma más conocida e inmediata de operar. Es como si le dijéramos al aparato: "toma el tres, multiplícalo por ocho y dame el resultado".

Pero este sistema operativo, en el que el operador se halla comprendido entre los operandos, no es el más eficaz. Habrán notado que al apretar el operador en las calculadoras corrientes el número que está en el display se apaga un instante. Lo que ocurre es que el microprocesador efectúa un control de memoria para comprobar que no hay en ella otra cosa más que un número (el último pulsado u obtenido). Todo ello se efectúa en pocos micro-

segundos, pero se trata, aun así, de un tiempo desperdiciado. Habrá observado también que con el resultado o con cualquier otra operación sucesiva esto no se produce.

Otros sistemas operativos utilizan una notación en la cual el operador precede a los operandos. Quienes conozcan el Lisp (si no lo conoce, un próximo volumen de la B.B.I. le dará la oportunidad de subsanar este fallo) reconocerán la expresión:

(sum(3 4))

Esto es lo mismo que si dijéramos "suma 3 y 4".

El lenguaje FORTH utiliza la notación RPN (Reverse Polish Notation), es decir, "notación polaca inversa", así llamada en honor del polaco Lukasievisz, que la inventó. En un principio tal vez resulte algo complicada, pero basta con un poco de práctica para que se convierta en la cosa más natural de este mundo.

Probablemente habrá tenido alguna vez entre las manos una calculadora de la firma Hewlett-Packard. Lo primero que le habrá llamado la atención es la falta de la tecla "=". Las calculadoras HP utilizan para sus operaciones un stack (pila, o pila operativa). Explicaremos ahora, sirviéndonos de un ejemplo, de qué se trata y cómo funciona esta estructura, que al ser clave fundamental en este lenguaje, seguiremos viendo y necesitaremos a lo largo de los restantes capítulos.

Podríamos comparar el stack a un depósito vertical móvil del mostrador de un restaurante, con un muelle que empuja hacia arriba los platos que guarda. Cada vez que se añade un plato el muelle cede un poco y toda la pila de platos desciende un puesto para admitirlo. En el caso contrario, si quitamos platos, el muelle se destensa y la pila sube el mismo número de puestos. Naturalmente, los últimos platos que pongamos serán los primeros en ser utilizados y viceversa. Esta estructura se llama de tipo LIFO (last in-first out, es decir, "primero en entrar último en salir").

Los operadores (como en el caso del anterior, la multiplicación) trabajan en RPN sobre datos que ya están introducidos en el stack; el símbolo de la operación sigue a los datos que debe manipular. Evidentemente tendrá que existir en estas calculadoras una tecla que nos permita meter los datos en el stack antes de proceder a la operación. Refiriéndonos de nuevo a la HP, la tecla para este fin es ENTER.

Resulta claro ahora por qué la tecla "=" no tiene sentido alguno. En notación polaca inversa el resultado aparece directamente en el display al apretar la tecla de operación. El ejemplo anterior se habrá convertido así en:

Secuencia	Display	Stack	
		Punto superior	Siguientes
3	3	3	
ENTER	3	3	3
8	8	8	3
×	24	24	

que es lo mismo que si hubiéramos dicho a la máquina: "te doy el 3 y el 8, multiplícalos".

Vemos que después del 8 no es necesario pulsar ENTER. Efectivamente, esta acción es sólo necesaria cuando queremos separar o distinguir dos números, pero aquí no hace falta, pues ya hicimos la distinción entre el 3 y el 8.

Pongamos ahora otro ejemplo: imaginemos que queremos sumar 3, 7 y 22. Podremos utilizar la secuencia

Secuencia	Display	Stack		
		Superior	Siguientes	
3	3	3		
ENTER	3	3	3	
7	7	7	3	
ENTER	7	7	7	3
22	22	22	7	3
+	29	29	3	
+	32	32		

En las dos últimas fases lo que sucede es: calcula 7+22 y pone el resultado en el stack; a continuación suma a esa cifra el valor que encuentra (3) y el resultado lo coloca también en lo alto de la pila.

Puede comprobar que en el display aparece siempre lo que se halla contenido en la posición (registro) más alta del stack. El display, como suele decirse, "señala" siempre hacia el registro que se halla más arriba ("Tos", Top of stack) visualizando su contenido.

Ahora bien, la misma operación vista antes puede efectuarse de otra manera:

Secuencia	Display	Stack	
		Superior	Siguientes
3	3	3	
ENTER	3	3	3
7	7	7	3
+	10	10	
22	22	22	10
+	32	32	

La secuencia que resulta más intuitiva que la anterior equivale a: "te doy 3 y 7; súmalos... Te doy ahora 22; súmalo a lo que has obtenido antes (que tienes en TOS)". Esta secuencia ofrece, además, la ventaja de que ocupa durante su ejecución sólo dos registros de memoria (en lugar de tres). Asimismo, las operaciones se efectúan en cadena, es decir que se utiliza cada vez el resultado parcial y la secuencia de teclas a pulsar es menor (6 frente a 7).

El ejemplo, por otra parte, nos indica que cuando efectuamos una operación el propio símbolo indicativo hace de ENTER entre el resultado que provoca y el siguiente número que introduzcamos.

¿Resulta de veras tan útil el RPN o es sólo cuestión de gustos y práctica? Cierto es que en lo referente al aspecto puramente manual y como ocurre tan a menudo, el camino ideal es el del medio. Por un lado resulta más sencillo plantear una operación en SOA, pero también es verdad que con él no se podrán obtener siempre los resultados parciales ni, por ejemplo, hacer comprobaciones en sentido inverso, sobre todo si hay paréntesis, en una o varias jerarquías.

El RPN posee la pequeña desventaja de que, a primera vista, no resulta muy familiar, pero bastará un poco de práctica, que se obtiene con mucha rapidez, para tener a nuestra disposición un medio extremadamente potente, que permite realizar comprobaciones parciales, cálculos en cadena con conservación de datos y utilización de constantes sin tener que recurrir a la memoria.

Por otra parte, el RPN resulta mucho más lógico y obliga a hacer un análisis suplementario del problema para decidir el mejor planteamiento de los datos que hay que meter, cosa que en el fondo no es negativa. Es, además, más rápido en su ejecución. Este factor, que posee poca importancia en relación con el tiempo necesario para pulsar las teclas, resulta fundamental a la hora de trabajar con programas ya escritos, en ejecución, en los cuales la habilidad manual del operador tiene poca importancia.

Pero el RPN resulta la mayor parte de las veces más cómodo y breve, incluso para cálculos manuales. Consideremos, por ejemplo, la siguiente operación:

$(3\times8)\times(5-2)-(6-4)$

En SOA sería necesario pulsar 18 teclas (17 y el "=" final) para efectuarla. Veamos, en cambio, qué ocurre en la notación polaca invertida (aparece también el contenido del stack con la convención de que el valor que se halla más a la izquierda representa el TOS, tal y como hasta ahora):

Secuencia	Stack
Comienzo	Stack vacío
3	3
ENTER	3 3
8	8 3
×	24
5	5 24
ENTER	2 2 5 24
2	2 5 24
−	3 24
×	72
6	6 72
ENTER	6 6 72
4	4 6 72
−	2 72
−	70

Para obtener el mismo resultado hemos apretado sólo 14 teclas; la conclusión es evidente, ¿no?

De los ejemplos vistos hasta ahora puede fácilmente deducir en qué orden se extraen los números para las operaciones. Recuerde siempre que para que un operador pueda ser aplicado es necesario tener algo en el stack y no olvide que el número situado en el TOS es el que se utiliza como sumando, sustraendo, multiplicador o divisor del siguiente.

CAPITULO IV

OPERACIONES ARITMÉTICAS

Algunas precisiones acerca de los números

En FORTH la unidad base de los datos está representada por un número. Pero en este lenguaje, lo que hemos definido de una manera tan sencilla está constituido por el conjunto de valores que pueden ser representados en 16 bits (dos bytes) de la memoria del ordenador, o, mejor dicho, en los 15 bits más a la derecha (de orden más bajo), al estar el bit más alto (most significant bit: MSB) reservado para el signo (generalmente 0=+ y 1=-).

Debido a que en binario el valor mayor representable en tal supuesto es de +32767 tendremos que el campo de valores que con dos bytes podremos representar irá desde +32767 hasta -32768.

A menudo (un ejemplo puede ser la longitud del lado de un polígono, la duración de un período de tiempo o la dirección de una locación de memoria) no nos será necesario utilizar el bit reservado para el signo, así que podremos usar todos los 16 bits. En este caso los valores que podrán ser asumidos irán desde 0 hasta 65535.

Estos números se llaman sencillos, con signo o sin él (unsigned=U) y, evidentemente, poseen un campo de variabilidad que no es muy elevado, sobre todo para las aplicaciones financieras y científicas. Charles Moore, en su artículo "The evolution of FORTH, an unusual language", publicado en agosto de 1980 en la revista Byte, cuenta que el primer problema de este tipo le surgió cuando empezó a hacer los cálculos de los gastos consiguientes a la apertura de la FORTH Inc.

Dejando de lado la anécdota (Moore, en su anterior vocación astronómica, suponemos que tuvo que vérselas con cifras mucho mayores) está claro que es fundamental poder disponer de números más elevados. Esto se consigue con los números dobles (señalados también aquí con la letra "D": (D=doublenumber) tal vez más conocidos por ser utilizados en otros lenguajes como números de doble precisión; ocupan 32 bits de memoria. También aquí los hay con signo (variabilidad de -2.147.483.648 a +2.147.483.647) y sin él (entre 0 y 4.294.967.295).
utilizan incluso números de triple y cuádruple precisión, pero en esta ocasión no nos referiremos a ellos.

Es de lo más sencillo de este mundo introducir los números en el stack: bastará con teclearlos y pulsar a continuación ENTER (RETURN, END LINE, CR según el ordenador). Cada vez que se incluya un número nuevo toda la pila bajará un puesto.

Las dimensiones de la misma varían según los sistemas, pero si tenemos en cuenta que incluso las calculadoras más potentes de HP, como la 41C cuyo stack poseen sólo cuatro posiciones, pueden efectuar los cálculos más complejos, se entenderá por qué sus dimensiones poseen relativamente poco interés, hasta el punto de que casi nunca se mencionan como dato relevante.

De todas formas cualquier stack, ya sea de 10 ó de 300 plazas, obedece a ciertas reglas fundamentales. Teniendo en cuenta que se trata de pilas con fondo cerrado (al contrario que las calculadoras de bolsillo, en las que cuando se añade a una pila de cuatro plazas, que son siempre con fondo abierto, un quinto valor, el primero que fue introducido se pierde), la inserción de más valores que lugares hay disponibles produce el mensaje:

STACK FULL

y ése y los valores posteriores serán rechazados.

Los números dobles se introducen en el stack de la misma manera que los sencillos, pero deben contener un punto decimal. Este puede hallarse en cualquier parte: así, 90000, para ser introducido en el stack, tendrá que teclearse como ".900000", "9.00000", "900000.", o como mejor le parezca.

Al trabajar en FORTH en un ordenador el resultado de la operación se sitúa en el TOS, no en la pantalla, cosa que no resulta nueva. Salvo raras excepciones, cualquier sistema sobre el que se esté operando no proporciona directamente los resultados de sus cálculos. Para que el resultado aparezca en pantalla o impresora habrá que indicarlo, como con el PRINT del BASIC. En FORTH esta operación se efectúa incluyendo un punto [.] en la secuencia en el momento en que se quiera visualizar el resultado.

El punto [.] posee también la capacidad de extraer el número del TOS, así que se pierde y no estará ya disponible para su utilización. Más adelante veremos de qué manera se resuelve este problema.

Para que un número doble que se halla en el TOS, con o sin signos, aparezca en pantalla, se requerirá un poco más de esfuerzo. Será necesario anteponer al [.] respectivamente las letras D y U: así tendremos [D.] y [U.].

Cuando el stack está vacío la operación de extracción determina condiciones de error, y se produce el mensaje

EMPTY STACK

Tan sólo una última precisión: los ordenadores más corrientes tienen procesadores de 8 bits. Esto quiere decir que incluso el número más sencillo ocupará dos bytes de memoria (o dos bytes de stack, naturalmente).

La manera en que está situado el número sencillo en estos dos bytes que ocupa depende tan sólo de la arquitectura general del ordenador. En algunos sistemas los ocho bits más altos se hallan en la locación de memoria más baja; en otros ocurre lo contrario. Como es lógico, con números dobles todo se duplica. Pero, en honor a la verdad, a menos que desee examinar la memoria bit por bit el asunto posee poca importancia. Será el procesador el que tendrá que vérselas con esos problemas. La figura 1 muestra de qué manera se hallan normalmente situados los números en el stack o en memoria.

Suma

Como cualquier otro lenguaje, el FORTH posee una gama de operaciones matemáticas básicas. La palabra [+], como es lógico, suma los dos números que se hallan en cabeza del stack y deja la suma algebraica en el TOS. Naturalmente, para que el resultado sea visualizado habrá que añadir el punto final. Así pulsaríamos:

```
3 8 + .
```

donde los espacios que introducimos marcan ya la separación entre los números y los operandos.

Al seguir esta secuencia con el RETURN se verá en la pantalla el resultado (11) y quedará el stack vacío. Aprovechamos la ocasión para precisar que de ahora en adelante daremos por supuesto que habrá que pulsar la tecla RETURN, a menos que el hecho de omitirlo pueda originar confusión.

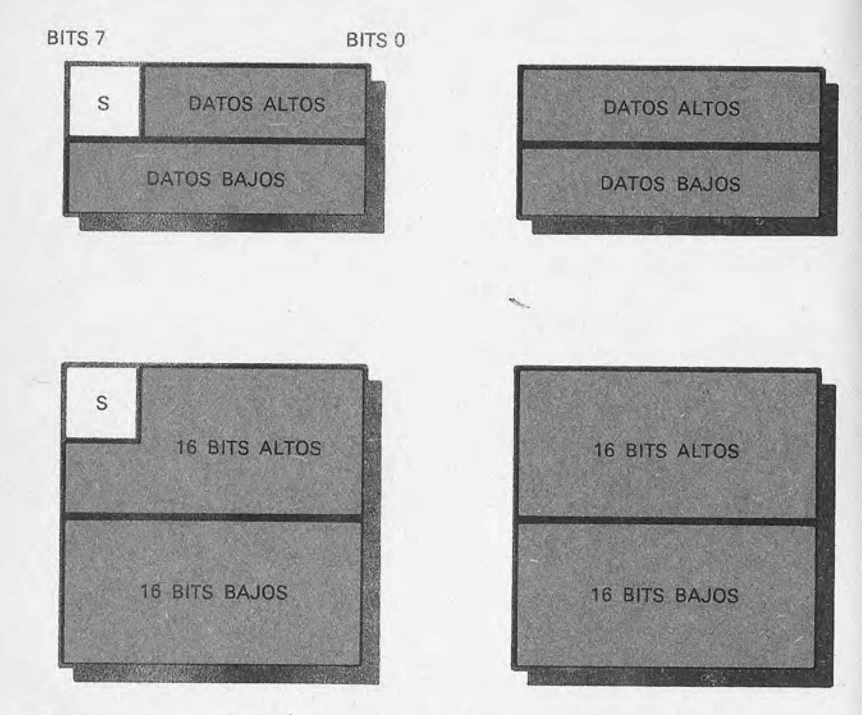


Figura 1.—Formas de memorizar los distintos tipos de datos.

Hay dos maneras de sumar varios números; por ejemplo, 1+2+3+4 puede efectuarse de la siguiente forma (repetimos que con el espacio entre los números estamos sobreentendiendo ya el ENTER):

1 2 + 3 + 4 +   10 OK

o también de esta otra:

1 2 3 4   + + +   10 OK

El OK final aparecerá siempre después de cada operación o secuencia cuando se verifique sin errores. Si los hay aparecerá un signo de interrogación seguido, generalmente, por un breve mensaje que informa acercad el tipo de error cometido.

Hay otras dos palabras de adición, [1+] y [2+] que añaden 1 ó 2 al TOS. Podrían parecer aparentemente superfluas, pero en

operaciones sobre la memoria o cuando no se quiera cambiar la posición de los números en el stack resultan muy útiles, ya que con ellas no se producen desplazamientos hacia arriba o hacia abajo dentro de la pila.

Los números dobles utilizan la palabra [D+], cuyo significado es evidente.

Resta

La operación opuesta se efectúa con la palabra [-] que, y cuidado con esto, resta el número contenido en el TOS del segundo número del stack y coloca en TOS la diferencia (a menos que se quiera visualizarla por medio de [.]).

Igual que con la suma, también en este caso existen las palabras [1-], [2-] y [D-].

Multiplicación

La palabra [*] permite multiplicar dos números en el stack; el producto se mantiene en el TOS. ¡Atención!: si no especificamos los tipos de números utilizados es posible caer en un error con facilidad. Por ejemplo, al multiplicar 30000 por 2 se producirían, dada la limitación de los números sencillos con signo, condiciones de error (60000 supera los 32767). Esto puede ser resuelto utilizando la palabra [U*], que permite utilizar el octavo bit (números sin signo), de tal forma que la secuencia

30000 2 U* . 60000 OK

dará el resultado correcto.

División

La división posee tres operadores diferentes que, en todo caso, permiten dividir el segundo número del stack por el contenido en el TOS.

El primero, el más corriente, es [/] que deja en el TOS el cociente truncado. Por ejemplo:

11 4 / .   2 OK

da el resultado efectuando el truncamiento (no el redondeo) de la operación.

La palabra MOD (módulo) hace la división igual que la anterior, pero deja en el TOS sólo el resto. La secuencia anterior se convierte en:

11 4 MOD . 3 OK

Nótese que el resto posee siempre el signo del dividendo.

Por último U/MOD divide un número doble por otro sencillo, dejando en el TOS el cociente y en segundo lugar el resto. Se trata, con alguna diferencia, del compendio de las anteriores.

De igual manera que lo que ya vimos, existe la palabra 2/, cuyo significado es evidente.

Jerarquía de las operaciones

Muchos lenguajes evolucionados poseen jerarquías en el desarrollo de una secuencia de operaciones. En la notación polaca inversa, naturalmente, debido a la sintaxis tan particular usada en la resolución de operaciones, esto no tiene sentido. Recuerde que los números que hay que manejar, del mismo modo que los platos que el cocinero coge de nuestro imaginario estante, parten siempre de lo alto.

Es por eso que algunas estructuras básicas aritméticas pierden aquí todo significado. Por ejemplo, en las calculadoras de bolsillo con RPN son del todo inexistentes los paréntesis y sus jerarquías. En FORTH éstos tienen una función totalmente distinta y nunca para procedimientos matemáticos. Si le resulta difícil razonar de esta manera y piensa que se trata de una complicación inútil, sepa que quienes se introducen en la RPN acaban por abandonar casi siempre la notación algebraica y utilizar en su lugar la RPN.

Otros operadores

Hay además otras palabras destinadas a manipular aritméticamente números presentes en el stack. Así [*/] calcula, del mismo modo que lo haríamos con papel y lápiz, la parte fraccionaria de un número; es necesario para ello poseer en stack tres valores (de abajo a arriba): la base, el numerador y el denominador de la fracción.

La palabra */MOD permite, además, conservar el resto.

Elevar un número a una potencia se efectúa con [**]. Por ejemplo:

4 5 **. . 1024 OK

NEGATE cambia de signo el TOS

66 NEGATE. . -66 OK

ABS proporciona el valor absoluto del mismo modo que DABS, que trabaja sobre números dobles.

Por último, el lenguaje FORTH, igual que otros muchos, permite comparar dos números, de tal forma que pueda elegirse el mayor o menor. Tendremos para ello las palabras MAX y MIN.

3 6 MAX . 6 OK
-3 -2 MIN . -5 OK

Formalismo representativo

Concluimos estableciendo un formalismo en la representación usada. Por convenio, las operaciones, palabras y secuencias operativas se representan en FORTH de la siguiente manera:

op. op. etc. - - - result. result.

donde el número de operandos y de resultados puede variar desde 0 hasta un número muy elevado; la secuencia "---" representa la(s) palabra(s) que hay que ejecutar. Por ejemplo, la multiplicación sería del tipo

n1 n2 - - - n3

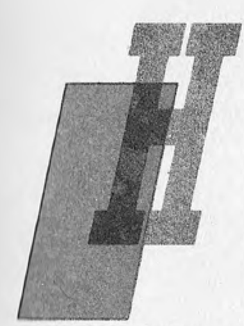
Esto significa que en TOS debe haber dos números (n1 y n2). El grupo "---" representaría la ejecución de la palabra (en este caso [*]) y n3 el resultado.

Una operación no tiene por qué dar necesariamente resultados numéricos; hay una palabra, por ejemplo, que limpia el stack y que no requerirá, como es obvio, ningún número determinado de parámetros antes de su ejecución y tampoco podrá, naturalmente, presentar resultados después de ella. Esta se podrá representar como

n1, n2, n3, ... nn - - -

CAPITULO V

MANEJANDO EL STACK



Hemos dicho que el FORTH está, sobre todo, basado en palabras (words) y ya hemos visto su funcionamiento como operadores aritméticos. Pero en el transcurso de un programa los datos en el stack no siempre se presentan en el orden deseado, sino que, siguiendo las leyes de Murphy, estarán siempre en el opuesto. Deben existir, por tanto, algunas palabras que tengan la capacidad de desplazar, reordenar, copiar en otro lugar, etc., los componentes del stack de tal forma que sea posible efectuar sobre ellos todas las operaciones.

En efecto, no hay que olvidar que este lenguaje debe, entre otras cosas, su rapidez al hecho de que utiliza la pila operativa, accesible al instante y de más fácil acceso a los datos contenidos en ella que a aquellos asociados a una dirección de variable.

Explicaremos a continuación cómo se efectúan estas operaciones y pasaremos después a ver de qué manera se construyen nuevas palabras.

Manipuladores del stack

Se denominan así porque actúan sobre el contenido del stack efectuando duplicaciones, desplazamientos, inversiones, etc. A pesar de no ser muchos permiten prácticamente cualquier operación en la pila; se trata de unas herramientas indispensables para definir las nuevas palabras que habrá que incluir en el diccionario básico del sistema.

La palabra de manipulación del stack más sencilla es DUP, que duplica el contenido del TOS desplazando hacia abajo la pila una posición. Se tendrá, por ejemplo:

n1 n2 - - - n1 n2 n3

o, de forma más clara:

Stack	DUP	Stack
5 (TOS)		5 (TOS)
3		5
2		3
...		2
		...

Hay otra palabra parecida, ?DUP (-DUP en FIG-Forth) que hace, antes de la duplicación, un test sobre el TOS, efectuando la duplicación del número sólo si éste es distinto de 0.

DUP, independientemente de su uso a lo largo de un programa, tiene una utilidad inmediata. Sirve, por ejemplo, para duplicar un número (DUP+), elevarlo al cuadrado (DUP*), el cubo (DUP DUP **), etc. Permite además remediar un defecto grave de la palabra [.] que, al ser ejecutada, borra del stack el valor visualizado. Por ejemplo:

3 8 * DUP . 24 OK

visualiza el valor contenido en el TOS pero conserva en el stack una copia del resultado que podrá ser utilizada más adelante. No representa un problema el que los operadores siguientes lo empujen hacia abajo en la pila: otras palabras, de las que hablaremos en breve, se encargarán de hacerlo emerger de nuevo.

La palabra 2DUP tiene el mismo significado para números dobles.

Si lo que necesitamos poner en el TOS es una copia, no del primero, sino del segundo número del stack, OVER (y 2OVER) efectuarán esta operación. Se produce, por tanto, el proceso que aparece en la figura 2.

PICK, una palabra que desgraciadmente no existe en FIG-Forth, es mucho más genérica y elástica: copia (cuidado, ¡no transfiere!) cualquier número presente en el stack al TOS. Como es lógico, se requerirá un parámetro que indique la posición del número que va a ser duplicado. Podremos representarlo de la forma

n - - - n1

Figura 2.—Secuencia de la palabra OVER.

donde el n1 es el número que se halla en el enésimo puesto del stack.

En realidad, el mismo OVER no es otra cosa que

2 PICK

DROP borra lo que se halla en el TOS y desplaza un lugar hacia arriba todos los números. En la práctica funciona como [.], pero no visualiza (o imprime) nada.

Hemos visto hasta ahora palabras que hacen una copia de números presentes en la pila. Pero puede ocurrir que lo que queramos sea tan sólo desplazar algunos números sin duplicar nada. Para ello se usan tres palabras que permiten tomar del stack un dato para ponerlo en el TOS. El puesto libre dejado por el número retirado hace bajar la parte anterior de la pila un espacio, de tal manera que su profundidad permanezca inalterada.

SWAP intercambia los dos números en cabeza de la pila.

ROT efectúa (Fig. 3) esta misma operación, pero esta vez con los valores primero y tercero.

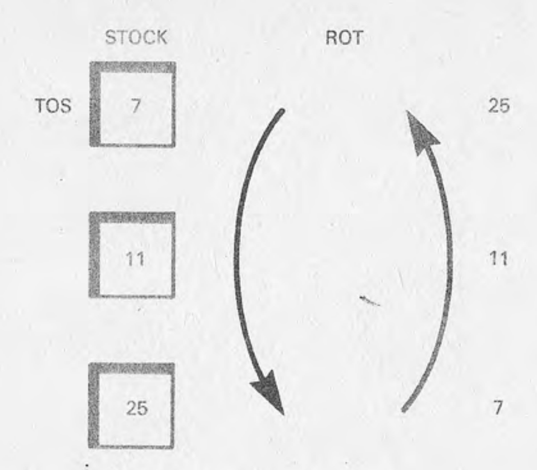


Figura 3.—Secuencia de la palabra ROT.

Para números dobles sirven también las palabras 2SWAP y 2ROT.

ROLL, que se halla sólo en FORTH-79 (y 83), permite el desplazamiento de cualquier número al TOS. Por ejemplo:

5 ROLL

llevará el valor que está en el quinto lugar de la pila al TOS, desplazando todos los demás hacia abajo.

Con toda esta historia de duplicar e ir de arriba a abajo es fácil perder la cuenta de cuántos números hay presentes en el stack y cuáles son. Para evitar esto existe la palabra DEPTH que proporciona el número de posiciones (de 16 bits) ocupadas en la pila. Evidentemente, para números dobles (32 bits) el valor será igual a la mitad. La secuencia formal es:

- - - n

que informa que hay «n» posiciones de 16 bits ocupadas en el stack, es decir n/2 números dobles.

Por último, el FORTH posee un indicador especial llamado "stack pointer" (puntero o apuntador de stack), que conserva la dirección del TOS. En cuanto se le reclama, el lenguaje activa una locación que contiene la dirección de la cima del stack. Cada vez que un número sencillo se incluye en el stack el indicador se de-

crementa en 2; por tanto contiene la dirección de la posición más elevada del stack.

Precisamente dos palabras específicas, 30 y SP@, pueden utilizarse para buscar la dirección del TOS y de la posición situada por debajo del último número presente en el stack. La figura 4 esquematiza el uso y las funciones de esta palabra.

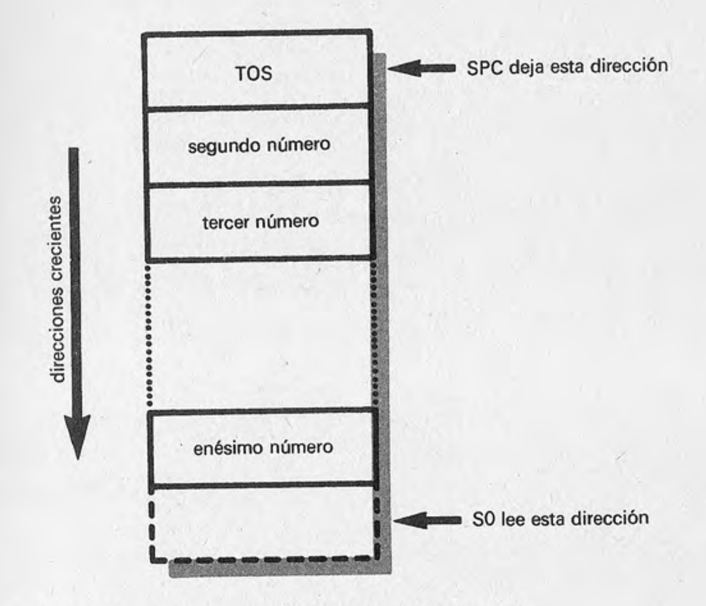


Figura 4.—Posiciones y direcciones del stack.

CAPITULO VI

CREANDO Y SUPRIMIENDO PALABRAS

Tal vez a más de un lector le haya surgido la tentación de no seguir adelante con el libro debido a las complicaciones formales que el FORTH presenta sin ofrecer claramente, al menos hasta ahora, nada ventajoso a cambio. Lukasiewicz, matemático polaco a quien debemos la notación RPN, cuando se le preguntaba el por qué de tantas supuestas dificultades en este lenguaje, respondía que lo que él había pretendido era no complicarse la vida con paréntesis, reglas jerárquicas y signos de igual, sin olvidar mencionar a este propósito la falta de iniciativa de quienes utilizaban el SOA.

Si siguen adelante con la lectura acabarán por comprender la utilidad y ventajas que el FORTH y la RPN presentan.

Un diccionario personalizado

Cualquier lenguaje, desde el BASIC al más complejo PL1, posee una serie de instrucciones, órdenes básicas que una vez tecleadas hacen que el ordenador ejecute algo. También el FORTH posee un vocabulario básico, más o menos numeroso según la implementación, pero que será siempre mucho menos voluminoso de lo que podríamos imaginar.

Uno de los aspectos que hace del FORTH un lenguaje único e inimitable es que resulta posible definir un vocabulario propio y hacer corresponder a la palabra que más nos guste una acción u operación.

Pongamos un ejemplo: imaginemos que tenemos en stack los números 4 y 6, que representan los lados de un rectángulo cuyo perímetro queremos calcular. Para ello tendremos que teclear:

```
4
6
DUP       duplica el 6
ROT       lleva el 4 al TOS
DUP       duplica el 4
+++       pone la suma en el TOS
. .       muestra el resultado
```

Podría utilizarse también cualquier otra combinación adecuada (por ejemplo: 4 6 2* SWAP 2* + .).

En este lenguaje es posible definir una nueva palabra que, cada vez que se utilice, tomará los dos datos que se hallen en stack como lados de un rectángulo y calculará su perímetro. Veamos cómo.

Establecemos en este caso que la palabra es PERIMETRO. Pulsemos:

```
: PERIMETRO (permite calcular el perímetro de un rectángulo
cuyos lados están ya en el stack).
DUP
ROT
DUP
+
+
+    ;
```

y acabamos por último con el RETURN. Parece que no ha ocurrido nada, pero si escribimos ahora "4 6 PERIMETRO" obtendremos un OK en pantalla, es decir, ha sido efectuada la secuencia de operaciones necesarias para calcular el perímetro del hipotético rectángulo y el resultado ha sido puesto en el TOS.

Este ejemplo sirve para analizar la síntesis de la secuencia tecleada: comienza con la palabra [:] que en FORTH avisa al sistema de que va a ser definida una nueva palabra. Sigue a continuación la expresión de la palabra en sí (nótese la separación de un espacio; espacios mayores que 1 serán ignorados); luego un paréntesis que indica la presencia de un comentario, cerrado por otro paréntesis (también éstos deben ir separados por un espacio) y, por último, la secuencia de operaciones necesarias para resolver el problema. La última palabra [;] avisa al sistema de que la definición ha concluido. En la práctica es como si hubiéramos

ordenado al ordenador que cada vez que escribamos PERIMETRO desarrolle exactamente esa misma secuencia de operaciones.

Como ven, poco a poco el tema va resultando algo más sabroso... Definiremos ahora otra palabra:

```
: AREA (calcula el área de un rectángulo)
*
  ;
```

y también:

```
: AMBAS (calcula perímetro y área de un rectángulo dados
los lados)
DUP  ROT  DUP (duplica los datos)
PERIMETRO (calcula el perímetro)
ROT  ROT (pone arriba los datos para calcular el área)
AREA  SWAP
."el perímetro es igual a"      . CR
."el área es igual a"           . CR
(fin definición)                ;
```

Vemos aquí un nuevo elemento, la palabra [."], que permite visualizar un mensaje terminado con ["].

Tecleemos

```
3  5  AMBAS
```

y obtendremos:

el perímetro es igual a 16
el área es igual a 15

seguidos por el consabido OK.

CR (carriage return) es una palabra que permite saltar a la siguiente línea. El stack, evidentemente, no contiene nada más.

Vamos a empezar ahora a hacer cosas un poco más interesantes. Si su ordenador no posee las palabras que calculan el cuadrado, el cubo o cualquier otra potencia será posible definirlas de la siguiente forma:

```
: CUADRADO  DUP          *;
: CUBO      DUP  DUP     **;
```

o también

```
:CUBO          DUP CUADRADO  *  ;
```

También puede elevarlo a la sexta potencia al teclear

: ALA6 CUADRADO CUBO

Podríamos compararlo a la construcción de un edificio utilizando ladrillos, vigas y tubos para hacer luego toda una ciudad al juntar varios edificios.

Los [:] efectúan también al ser usados otra serie de operaciones. En realidad, si pudiéramos "abrir" un lenguaje FORTH descubriríamos que posee una estructura de lo más desconcertante. En primer lugar es extremadamente compacto, funciona de una manera intermedia, resultando como una especie de unión entre un intérprete y un compilador, con una mezcla en principio extraña de secuencias en Assembler y FORTH.

Se trata a la vez de un sistema operativo, un monitor, un editor y un conjunto de utilidades (escritas también en FORTH) que constituyen el diccionario. A pesar de ello en cuanto se carga está representado nada más que por el stack, un intérprete y el diccionario. Y todo ello a menudo se halla contenido en unos cuantos Kbytes (algunos de los más potentes que hay en el mercado no alcanzan los 10 Kbytes) sin disminuir nada por ello en cuanto a operatividad, elasticidad y potencia se refiere.

Si no lo ha entendido todo acerca de la estructura del FORTH no vaya a preocuparse, pues en realidad no resulta nada sencillo. Por otra parte, el 99,9% de quienes programan saben muy poco acerca de cómo actúan las órdenes que están dando al ordenador. Y, al fin y al cabo, para conducir tampoco es estrictamente necesario saber de qué manera está hecho el motor en todos sus detalles...

Bastará con saber que cada FORTH incluye dos programas que se encargan de transformar las combinaciones de caracteres tecleados en secuencias que la CPU puede interpretar y ejecutar.

El primero de estos programas es el intérprete; transforma y efectúa cada orden recibida; ésta, una vez ejecutada, se olvida y el intérprete se para y espera otra nueva. Esto es lo que ocurre cuando se utiliza el ordenador como calculadora o también con palabras complejas que, lo recordamos una vez más, equivalen a "programas" en otros lenguajes.

El compilador, al contrario, no ejecuta inmediatamente la orden sino que recuerda los extremos y conserva en memoria sólo la dirección del resultado. La palabra [:] es la que se encarga de ponerlo en marcha "despertándolo" y señalándole que se quiere iniciar una compilación. La siguiente palabra que hemos elegido es el símbolo (token), es decir, algo así como la llave, la contraseña, y hará que se ejecuten las operaciones sucesivas. La palabra [;] indica, por último, la conclusión de las operaciones. A partir de ella se ejecuta la compilación que, si es correcta, irá seguida por el consabido OK o, en caso contrario, por "?".

Una vez compilada la nueva palabra entra a formar parte del diccionario. Incluso si se pierde su definición en código fuente (es decir, la secuencia comprendida entre [:] y [;]) no tendremos un problema demasiado grave: dentro de un poco aprenderemos de qué manera conservarla tal vez para modificarla y utilizarla en otra ocasión. De todas formas siempre es posible controlar las palabras presentes en el diccionario, tanto las predefinidas por la casa constructora como las que nosotros hubiéramos definido; bastará con teclear

VLIST

Al contrario del stack, que parte desde la posición de memoria más baja, el diccionario lo hace desde la dirección de memoria más alta. Por tanto VLIST comenzará desde la última palabra compilada e irá marcha atrás hasta la primera implementada por el constructor. Algunos sistemas proporcionan también a la vez la dirección.

En nuestro caso, VLIST dará

ALA6
AMBAS
AREA
PERIMETRO
.
. (palabras precedentes).
.

A veces puede ser necesario borrar del diccionario alguna palabra que ya no queremos conservar por ser inútil o errónea. Usaremos entonces FORGET, que va acompañada del nombre de la palabra que hay que eliminar. Pero hay que precisar algo: FORGET borra no sólo la palabra mencionada, sino también todas aquellas definidas y compiladas posteriormente. Así:

FORGET AREA

borrará AREA, pero también AMBAS y ALA6. Más adelante explicaremos por qué se produce esto (veremos que es, incluso, necesario); por el momento podremos tan sólo poner un ejemplo.

Ya hemos visto que el sistema operativo es de tipo acumulativo, es decir, junta cosas para formar otras nuevas. Si se produjera la eliminación tan sólo de la palabra AREA, AMBAS contendría una palabra sin significado (AREA precisamente), lo que provo-

caría un error en el sistema. De esta forma FORGET, tal y como funciona, está verdaderamente a prueba de error, al menos bajo este punto de vista. Veremos después cómo esquivar este problema...

El programa compilador es tan potente que permite también volver a definir una palabra si se desea (algunos sistemas poseen una estructura de control contra definiciones múltiples). En efecto, el sistema operativo no diferencia entre palabras ya presentes y aquellas añadidas al diccionario. En el momento en que se crea una nueva definición el ordenador se da cuenta de ello y avisa al usuario con un mensaje.

Si quisiéramos volver a definir CUADRADO para que calculara, por ejemplo, en lugar de la segunda potencia, la mitad del valor en TOS y tecleáramos

```
: CUADRADO 2 / ;
```

tendríamos un aviso del tipo

```
CUADRADO NOT UNIQUE OK
```

El OK final indica que, de todas formas, la nueva definición ha sido aceptada. La anterior permanecerá en el diccionario, aunque inutilizada, pues la búsqueda del valor y significado de las palabras se efectúa a partir de la última definición.

Si escribimos

```
FORGET CUADRADO
```

se borrará sólo la última definición de CUADRADO y, naturalmente, todas las palabras definidas después de ésa, de modo que

```
3 CUADRADO
```

producirá como respuesta

```
9 OK
```

vuelve a dar valores congruentes con la definición precedente.

Cada palabra, entonces, puede ser redefinida sin ningún problema. Puestos a imaginar podemos incluso cambiar el significado de los símbolos de operación. Se puede, por ejemplo, invertir el valor del signo + con

```
: + - ;
```


El sistema nos enviará el mensaje que indica la duplicación de una palabra. Al escribir seguidamente

```
4 6 +
```

obtendremos como respuesta

```
-2 OK
```

Naturalmente, por el momento no es el caso de llegar a tanta sofisticación (sobre todo porque nuestro equilibrio mental podría resentirse); tome esto simplemente como información para demostrar la potencia del sistema operativo.

Hay, además, otra palabra muy útil que permite conocer si una dada existe o no en el diccionario. Se llama "tick" y está representada por un apóstrofe ('). Veamos cómo se utiliza

```
' nombre de la palabra
```

Deja la dirección de la palabra requerida en el TOS. Por ejemplo:

```
' + .
```

proporciona (en este caso en un microordenador QL dotado de un FORTH de la Computer One):

```
1950 OK
```

1950 es la dirección en memoria de la palabra [+]. ¡Cuidado! Algunos sistemas, en el caso de palabras no definidas, dan una señal de error, generalmente del tipo "... non defined" borrando luego el stack (en ocasiones dan a continuación otro mensaje del tipo " ...? "). De modo que, para evitar riesgos de este tipo, es preferible utilizar VLIST.

Por último, una palabra puede ser cambiada de nombre, además de redefinida. En el primer caso el diccionario contendrá dos palabras con la misma función; en el segundo, en cambio, será una sola pero con dos definiciones distintas y, naturalmente, se aplicará sólo la más reciente.

La técnica de cambiar el nombre de una definición se utiliza generalmente para hacer compatibles programas escritos en dialectos diferentes. Por ejemplo, la palabra ?DUP en FORTH-79 duplica el TOS sólo si éste es distinto de cero. En FIG-FORTH tal función es efectuada por -DUP. Antes de utilizar un programa en FIG que contiene -DUP en un sistema compatible FORTH-79 será necesario añadir al diccionario la definición

```
: -DUP ?DUP ;
```

¿Va haciéndose cargo de la potencia y flexibilidad del FORTH?

En el momento en que se procede a definir y compilar nuevas palabras resulta muy útil, y a veces obligado, introducir de vez en cuando en el vocabulario unos puntos fijos, de, digámoslo así, orientación, que permitan subdividir las nuevas palabras recién introducidas en función, por ejemplo, de su significado y uso. Se trata de una especie de marca que puede resultar útil en ciertos casos. Esto puede hacerse por medio de palabras que no hacen nada del tipo

```
: word ;
```

Puede usar un nombre cualquiera, incluso tal vez seguido de un número (¡cuidado con la longitud máxima admitida por el sistema!). Así tendremos, por ejemplo

```
pepe 1
...
...
...
pepe 2
...
...
...
pepe 3
...
...
...
pepe 4
...
```

De esta forma es como si subdividiéramos el diccionario en capítulos, que harán más fácil de leer el VLIST. Será también posible usar el FORGET de una manera más orgánica y eficaz.

CAPITULO VII

MEMORIA, BLOQUES Y PÁGINAS

Cómo conservar lo que nos interesa

Cualquier palabra definida y compilada puede utilizarse mientras permanezca en la memoria, es decir, mientras no la borremos o no se desconecte la fuente de alimentación. Cierto es que un ordenador valdría muy poco si hubiera que teclear de nuevo todas las instrucciones de los programas después de cada encendido. Todo lenguaje dispone de instrucciones para cargar sus programas en algún dispositivo de grabación (mass storage unit) y el FORTH no podía ser menos, de modo que existirán también palabras para las operaciones de carga, grabación, modificación y supresión de datos y programas en la memoria de masa, que puede estar constituida por disquetes (discos flexibles) o cinta.

Hay un método que algunas personas emplean para conservar las palabras que han definido, pero en honor a la verdad no es muy bueno: como a menudo el lenguaje se proporciona en un disquete como si de un programa en código máquina se tratara bastará, al final, con salvar todo el lenguaje en disco para poder disponer más tarde nuestras propias palabras listas para la próxima carga. Pero esto resulta poco eficaz, ya que incluso tras pocas sesiones el lenguaje habrá acumulado tantas palabras (inútiles las más) que resultará inoperante debido a la saturación de la memoria.

Por otra parte, es siempre conveniente conservar una copia de las palabras propias en su código fuente (es decir lo que está definido entre las palabras [:] y [;]), para poder efectuar más tarde si lo deseamos modificaciones, uniones o incluso para recordar y

entender las funciones y el significado de las diferentes palabras. En efecto, normalmente cada paquete FORTH contiene un programa editor destinado a modificar los textos y programas ya introducidos.

A pesar de que es posible utilizar un ordenador dotado de lenguaje FORTH sin poseer unidad de memoria de masa, se convierte entonces en poco más que una calculadora. En FORTH, de todas formas, la unidad de disco (o de cinta: de ahora en adelante siempre que mencionemos una estaremos incluyendo también la otra) se utiliza de una forma diferente que en otros lenguajes.

En FORTH un disquete puede compararse a un conjunto de casillas postales que, dispuestas ordenadamente y numeradas, se llaman bloques. Estos forman la unidad principal de transmisión de datos entre la memoria de masa y el ordenador; constan de grupos de 1024 bytes. Naturalmente, el número de bloques que hay en cada disco depende de la capacidad (caras, densidad) de la unidad en sí. Por ejemplo, un disquete de 5 ¼" de 13 sectores contiene 110 bloques.

La palabra bloque se refiere generalmente a la transferencia de datos a código máquina, pero los disquetes pueden utilizarse también para contener programas en código fuente. La unidad principal, que comprende también un Kbyte (1024 bytes), se denomina en tal caso página. Habrá, por tanto, palabras que permitan compilar una página y otras ejecutar un bloque. A veces la palabra en sí se comportará de forma diferente según uno u otro caso.

Las páginas están generalmente organizadas en grupos de 16 líneas de 64 columnas (16×64=1024). Pero esto no es una regla, pues puede haber líneas de 40 caracteres y páginas de 24 líneas.

Cada página o bloque reclamado por la unidad de memoria de masa o destinado a ella permanece en un área particular de memoria, llamada buffer, cuya amplitud es de 1024 bytes. El número de buffers que hay en un sistema puede ser de lo más variado, pero habrá al menos un par, incluso en las implementaciones más modestas. Una de sus funciones es obvia; reducir los tiempos de acceso a la memoria de masa durante el procesamiento, pues tales operaciones son más bien lentas, por cuanto dependen de dispositivos mecánicos como las unidades de cinta o de disco.

Las relaciones existentes entre los buffers, la memoria de masa, la pantalla y el teclado aparecen en la figura 5. Por suerte resulta bastante raro que el usuario necesite tener en cuenta cuál es la situación de los buffers en cada momento. De ello se encargará el sistema operativo que, en caso de necesidad, efectuará la operación que se requiera y dará el mensaje más conveniente.

Cada buffer está formado por un área de memoria principal, un identificador que contiene el número de bloque (o página) y

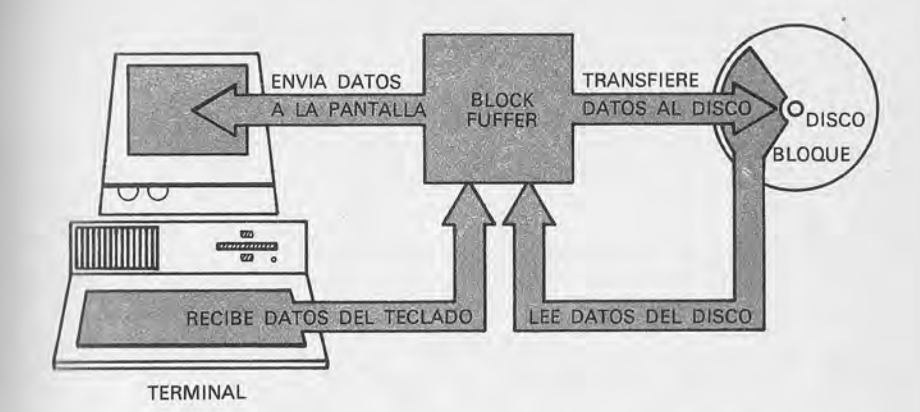


Figura 5.—Correlaciones entre memoria de masa, buffer, teclado y sistema de salida.

una bandera o identificador ("flag") que señala si el contenido del buffer ha sido modificado o no. Cuando el usuario pide que se cargue un bloque en memoria, el sistema operativo inicia la búsqueda por medio del identificador; si ese bloque se halla ya en memoria enviará un simple mensaje de OK. Si, por el contrario, reside aún en disco verificará el estado del buffer más próximo. Si la bandera está puesta a 1 (true), se transfiere su contenido al disco y carga el nuevo bloque en ese buffer. Como puede verse, la verdadera diferencia entre sistemas con distintas capacidades de buffers se debe a la mayor o menor necesidad de transferir datos al disco, lo que comporta al final una pérdida de tiempo más o menos grande.

Para escribir un programa en FORTH se siguen nueve fases principales:

- vaciar los buffers;
- seleccionar una página;
- ejecutar el programa editor;
- preparar las definiciones de las nuevas palabras;
- transferir el texto a la página. A menudo estas tres últimas fases están combinadas y representan, en la práctica, una sola fase formal o sustancial;
- conservar el código fuente redactado en disco;
- compilar el código fuente. Generalmente esto se efectúa reclamándolo directamente del disco; a veces también se utiliza el programa ya residente en memoria;

- ejecutar el programa y comprobar su validez y funcionalidad;
- concluir la sesión de trabajo o, en caso de error, dar cuenta de él y volver al paso 3 para su corrección.

Vaciar los buffers

Es el primer paso en la realización de un programa. Se devuelve el ordenador al estado de "limpieza" inicial. La palabra adecuada es:

EMPTY_BUFFER

Selección de una página

Esto se efectúa con la secuencia:

n SCR ! - - -

La variable SCR contendrá el número de la página utilizada; todas las operaciones de I/O siguientes harán referencia a ella. La página 97 puede ser llamada con

97 SCR !

Utilización del editor

Las tres fases siguientes (llamar al editor, preparar las definiciones, transferirlas a la página) varían completamente de sistema a sistema. Se trata, en realidad, de efectuar las operaciones comunes a todos los lenguajes, como seleccionar y escribir líneas, corregir aunque sólo sean los errores cometidos al pulsar las teclas, mover el cursor, borrar definiciones enteras, parte de las mismas, etc. Para ello, a pesar de que hay un estándar del FORTH Interest Group, cada constructor da más importancia a ciertas fases de la edición que considera de mayor utilidad. Por este motivo es muy importante leer atentamente el capítulo del manual relacionado con tales operaciones.

Una vez concluidas las operaciones de edición es conveniente avisar al sistema de que la página redactada contiene informaciones que hay que transferir al disco en la primera ocasión que se presente. Esto se realiza con la palabra

UPDATE

y, por regla general, se produce automáticamente al final de cada sesión de edición o cuando una página está llena.

Escritura de una página en disco

Ahora hay que salvar el código fuente en disco en correspondencia con la página solicitada por la variable SCR. Esto equivale al SAVE o STORE de otros lenguajes y se obtiene con la palabra

SAVE_BUFFER

(FLUSH en FIG-FORTH). Tal comando no requiere parámetros puesto que cada buffer contiene ya un número de página-bloque.

Introducción del código objeto en el diccionario

El texto, o código fuente, está ya presente en disco. Para que las palabras que se hallan en él sean implementadas en el sistema (y en el diccionario) hay que ejecutar la palabra

n LOAD

Por ejemplo

97 LOAD

compila las palabras de la página 97 y las incorpora al diccionario exactamente igual que si fueran tecleadas y puede utilizarse de la misma manera. Nótese que en este caso la presencia en la memoria central del buffer que ya contiene la página 97, evita operaciones en disco. En caso contrario la página se carga desde el disco y es inmediatamente compilada.

A veces puede ser necesario, sobre todo en fases de depuración, aportar modificaciones al código fuente. Para ello bastará con escribir

102 LIST

que dará un listado del programa en la pantalla, llamando a la vez al editor, el cual permitirá que sean efectuadas las operaciones de modificación, corrección y añadido necesarias.

Examinemos un programa

El programa, en código fuente naturalmente, está representado por 16 líneas, numeradas de 0 a 15 y que contienen las instrucciones destinadas a resolver un problema determinado. Vamos a referirnos a la figura 6. La primera línea contiene un comentario; a

0	(CALCULO DEL IVA)
1	(EL VALOR DEL IMPORTE ESTA EN TOS)
2	: IVA (DEFINE UNA NUEVA PALABRA)
3	DUP DUP
4	12 100 */ (RECUERDE MULTIPLICA Y LUEGO DIVIDE)
5	DUP ROT
6	. "EL IVA, AL 12% SOBRE UN VALOR INICIAL DE "
7	.
8	. "ES DE "
9	.
10	+ (SUMA EL IVA AL VALOR BASE)
11	. "PTAS, LO QUE SUPONE UN VALOR TOTAL (IMPONIBLE MAS IVA) DE "
12	.
13	(FIN DEFINICION)
14	;
15	
16	

Figura 6.—Listado de la página "CALCULO DEL IVA".

pesar de que la organización de la página es libre se suele introducir en esta primera línea un comentario que dé información sobre el contenido de la página. La segunda recuerda que el stack debe contener el valor inicial del importe. La línea 2 (tercera) define la nueva palabra, IVA (naturalmente, puede ser llamada como se quiera). Las líneas 3, 4 y 5 efectúan operaciones, duplicaciones y trasposiciones en el stack. Desde la línea 6 en adelante se encuentra la verdadera salida (para que los mensajes en la pantalla estén más ordenados es aconsejable poner en las líneas 7, 9 y 11 un CR (carriage return), es decir, un "salto a la siguiente línea").

Como puede, ver los comentarios (totalmente ignorados por el compilador) abundan, cosa que no es mala idea ya que los programas en FORTH no son por sí de lo más claros precisamente. La utilización del programa es clara:

```
20000 IVA
```

dará

```
EL IVA, AL 12% SOBRE UN VALOR INICIAL DE 20000 ES DE
2400 PTAS., LO QUE SUPONE UN VALOR TOTAL (IMPONIBLE
MAS IVA) DE 22400
OK
```

Puede ocurrir, y pasa a menudo, que un programa ocupe más de una página. La técnica normal de carga desde el disco impondría, si tuvieran que cargarse las páginas 50 a la 53, la pulsación de:

```
50 LOAD 51 LOAD 52 LOAD 53 LOAD
```

o, lo que es lo mismo:

```
50 51 52 53 LOAD LOAD LOAD LOAD
```

Otra técnica mejor podría ser definir en la primera de las páginas que hay que cargar una palabra que cargue los siguientes. Por ejemplo:

```
: CARGA 51 52 53 LOAD LOAD LOAD ;
```

pero ésta tampoco es la mejor manera de hacerlo. El diccionario estándar del FORTH contiene en cualquier dialecto, la palabra

```
-->
```

que, si está situada en la última línea de una página, avisa al sis-

tema operativo para que se ocupe de cargar y compilar la página siguiente.

Así que si las páginas 50, 51 y 52 contienen en la última línea la palabra --->, bastará con escribir

50 LOAD

para que las restantes, desde la 51 hasta la 53, sean cargadas e interpretadas.

Hay además otra palabra:

;S

que detiene la interpretación de la página, a menudo al final de una secuencia [--->]. Se puede utilizar para separar los comentarios al final de una página y ahorrar así la fracción de tiempo necesaria para reconocer éstos.

Ya dijimos que la primera línea de una página se reserva generalmente para hacer comentarios sobre las funciones de la misma. Esto no es una costumbre o formalidad sin fundamento. En efecto, existe la palabra INDEX que muestra la primera línea de un grupo específico de páginas. Por ejemplo, la secuencia

22 25 INDEX

visualiza en la pantalla la primera línea de las páginas 22, 23, 24 y 25. Es evidente lo útil que resulta INDEX para buscar páginas cuyo número no recordamos. El FIG-FORTH posee además la palabra FENCE (se trata en realidad de una variable reservada; ya veremos más adelante de qué se trata) que, utilizada en la forma

' xxxxx FENCE !

ofrece una especie de seguro, ya que tanto la palabra xxxxx como todas las anteriores no podrán ser borradas del diccionario. Naturalmente, existe también la función opuesta, representada por la secuencia

0 FENCE !

CAPITULO VIII

TODO TIENE UN PRECIO: MATEMÁTICAS EN COMA FIJA

A pesar de cuanto hemos dicho en la introducción, resulta impensable que todo lo mejorcito aparezca reunido y sin inconvenientes, ya que, como todo el mundo sabe, no hay rosas sin espinas. Vamos a tratar ahora esos temas que, si bien no son puntos realmente débiles, sí resultan, al menos, controvertidos.

La que aparece como parte del título de este capítulo es, sin duda alguna, una de las cuestiones que recibe las mayores críticas y, por otro lado, las más enconadas defensas por parte de los usuarios del FORTH. Veamos de qué se trata y cuáles son los motivos de tales discrepancias.

Hagamos en una calculadora de bolsillo la siguiente operación:

2.50×3.31−1.175

Valores introducidos	Números en el display
2.50 ×	2.5
3.31 −	8.275
1.175	1.175
=	7.1

Como puede verse, el punto "fluctúa" según los valores decimales que la calculadora debe representar. Esta representación de los números, en la que no tiene importancia la cantidad de ci-

fras decimales de los operandos, se llama precisamente notación en coma flotante.

Al efectuar

0.04 / 2000000

tendremos, en calculadoras en las que no pueden aparecer más de 8 cifras, el resultado

2E-8

es decir, el resultado se conservará en memoria como si de dos números se tratara: "2" y "-8".

La utilidad de tal notación consiste en que es posible almacenar una enorme gama de números, desde el más pequeño valor hasta las medidas astronómicas más grandes utilizando, aunque sea a costa de una cierta aproximación, tan sólo dos números. Si tiene en cuenta que algunos ordenadores, incluso micros domésticos, son capaces de manejar valores entre E+500 y E-500 (la distancia entre la Tierra y el Sol es de 1.44E8 Km) podrá comprender lo inmensamente grande que resulta este campo de trabajo.

La notación en coma fija, en cambio, es el método para conservar valores en memoria sin variar la posición de la coma. Por ejemplo, si trabajamos con medidas en metros y centímetros todos los valores deberán ser reducidos a este submúltiplo.

Imaginemos que queremos hallar el área de un rectángulo cuyas dimensiones son 3.25 m y 2.25 m. La secuencia siguiente muestra cómo están organizados los datos en las dos operaciones diferentes:

Operaciones	Coma móvil	Coma fija
3.25	325 (-2)	325
×		
2.2	22 (-1)	220
=	7.15	71500
Resultado	[presente en memoria como 715 (-2)]	

Es decir: en coma fija se utilizan todos los valores tras haberlos "referido" a la unidad más pequeña o, lo que es lo mismo, todos los valores se utilizan como si fueran enteros.

Consideremos ahora el siguiente ejemplo: hay que efectuar

77/13

cuyo resultado es periódico simple (5'923076923076...), al ser ambos números primos entre sí y el divisor distinto de 2 y 5 o múltiplo de éstos. Queremos aproximar el resultado a las centésimas; con coma fija, deberemos prever dos puestos decimales efectuando

7700/13

que dará 592; y esta cifra, puesta de nuevo en orden (digámoslo así), dará el resultado con los dos decimales que queríamos.

Posiblemente algún lector se esté haciendo la siguiente pregunta: ¿es realmente necesario complicarse la vida con todos esos problemas de reducción y de comas decimales cuando resulta tan sencillo teclear los números y esperar los resultados?

En realidad, la notación en coma flotante es más fácil y cómoda para el usuario, sobre todo cuando el programa tiene que manejar ecuaciones complejas cuyos resultados a menudo no son, en términos de valores absolutos, previsibles. Sin embargo, el programador en FORTH utiliza el ordenador de la forma más eficaz posible, y así debería ser para cualquier otro lenguaje. El problema radica en que los tiempos de manipulación de números enteros son mucho más reducidos que aquellos con números decimales (pueden llegar hasta 1/6). Además, al comprador de un ordenador que debe resolver problemas complejos le parecerá más importante la rapidez de ejecución y velocidad de los resultados que la facilidad de programación. Así, pues, como siempre, todo es relativo y depende, en definitiva, del uso que usted vaya a darle.

No hay que olvidar tampoco que el FORTH es un lenguaje destinado a operaciones en tiempo real y no hay otro más veloz que no sea el Assembler-lenguaje máquina, mucho más complejo y pesado de utilizar. Por otra parte, el FORTH fue destinado inicialmente a medidas de tiempo, controles automáticos, adquisición automática de datos y, en general, operaciones que requieren una respuesta inmediata a una secuencia de entrada.

Por cierto, ¿sabían ustedes que las tomas de "La Guerra de las Galaxias", con Harrison Ford en su cápsula espacial y Obiwan con su espada láser, han sido filmadas con cámaras guiadas por programas en FORTH? Parece evidente que algunas cosas deben sacrificarse en favor de la rapidez, a costa incluso de un mayor esfuerzo por parte del programador. En Assembler resultaría mucho más penoso, mientras que el lenguaje FORTH alcanza casi to-

dos sus objetivos sin ser en realidad, como parece, tan complejo de utilizar.

Hay también otra solución: la presencia de un chip que se dedicara tan sólo a ejecutar cálculos en coma flotante. Esta solución resulta eficaz y seguramente podrá resolver de forma definitiva el problema. El mismo Moore ha integrado a menudo en sus ordenadores estos procesadores, lo que indica la utilidad y validez de tal solución. El problema es que, si bien hay equipos destinados desde un principio al FORTH que han sido construidos con esta integración, no podemos decir lo mismo del restante 99 por 100 de las existencias, que poseen el FORTH implementado en disquete o en tarjeta, aunque es cierto que en este último caso el chip "matemático" puede a veces ser "simulado".

Hay también casos rarísimos en los que ha sido prevista la coma flotante, pero se trata de algo forzado y la velocidad desciende enormemente.

CAPITULO IX

EL CICLO ITERATIVO (DO-LOOP)



Hasta ahora hemos tratado de palabras que ejecutaban instrucciones de una forma secuencial. La palabra PERIMETRO que definimos en el capítulo 6 permitía calcular el perímetro de un polígono dados sus lados. El programa se ejecutaba en manera secuencial, es decir, el sistema calculaba la primera operación que aparecía en la definición, luego la segunda, la tercera..., hasta encontrar la palabra que implica la conclusión de la definición [;].

Surge a veces, sin embargo, la necesidad de efectuar una misma operación repetidas veces. Si tuviéramos que hacer esto con las palabras que hemos visto hasta ahora se haría muy pesado y con algún que otro inconveniente. Pongamos un ejemplo: vamos a suponer que queremos la lista de los 10 primeros números que aparecen en el stack; definimos entonces la palabra

```
: DIEZPRIMEROS
.CR .CR .CR .CR .CR
.CR .CR .CR .CR .CR;
```

Pero esta palabra ha sido definida de tal forma que resulta prácticamente inútil, además de muy rebuscada (¿qué pasaría si los números que queremos fueran 40, 100 o aún más?) y sin elasticidad alguna.

Por suerte el FORTH, al igual que otros muchos lenguajes, realiza esta función si escribimos la operación una sola vez y la metemos en un bucle (loop en inglés: anillo, ciclo) para que sea ejecutada un número «n» de veces.

La estructura fundamental de un ciclo LOOP es

valor final+1 valor inicial DO ..(palabras).. LOOP

La secuencia anterior será entonces

```
:DIEZPRIMEROS
11 1 DO . CR LOOP;
```

Esta misma definición puede ser aún más elástica si hacemos que los límites del DO-LOOP no sean números fijos comprendidos dentro de la definición en sí, sino otros tomados del stack. De tal manera

```
: IMPRIME-N DO . CR LOOP;
```

es una definición más universal y flexible, puesto que extrae del stack los valores límite del bucle. Así

```
11 1 IMPRIME-N
```

funcionará de la misma manera que DIEZPRIMEROS.

Es conveniente recordar de nuevo que cuando el intérprete FORTH encuentra los números 11 y 1 los carga en ese mismo orden en el stack. Es precisamente por este motivo que en las definiciones de estructuras LOOP que hemos visto el valor final se indica antes del inicial.

Tan sólo una aclaración más, con su ejemplo correspondiente. Esta vez queremos saber los cubos de los números del 15 al 7. En el capítulo 6 definimos ya la palabra CUBO (DUP DUP ** o también DUP CUADRADO *) suponiendo, claro está, que el sistema no posee la palabra [**]. Tendremos:

```
: YY DO CUBO . LOOP ;
```

que para

```
16 7 YY
```

nos dará

```
3375 2744 2197 1728 1331 1000 729 512 343
```


Pero podemos hacer ahora un trabajo "más fino" si definimos la siguiente palabra:

```
: YYz                 (calcula el cubo de los números com-
                      prendidos entre los valores que hay en
                      el stack)
DO
  DUP                 (duplica el número para obtener una co-
                      pia del valor del cual se calcula el cubo)
  CUBO                (cálculo del cubo)
  SWAP
  ."el cubo de"..."es".  (imprime el valor de la raíz y del cubo)
  CR                  (a la línea siguiente)
LOOP
;                     (fin definición)
```

Probablemente más de un lector se habrá dado cuenta de que el valor final queda siempre aumentado en 1. ¿Por qué ocurre esto? Vamos a tratar de explicar de qué manera funciona la secuencia DO-LOOP (Fig. 7). El FORTH lleva la cuenta inicializando un índice (una posición de memoria particular o contador) con el valor inicial y añade 1 cada vez que encuentra la palabra LOOP; evidentemente, después de haber pasado la primera vez por ella el contador vale 2 y como la salida del bucle se produce cuando el valor del índice es igual o superior al valor de fin de bucle es necesario que el valor final sea, precisamente, superior en uno a la diferencia final-inicio que deseamos realizar. Puede hacerse esto mismo reduciendo en 1 el valor de partida.

Los valores inicial y final pueden asumir, dentro de los límites de la matemática en coma fija (a menos que se posean sistemas con coma flotante), cualquier valor numérico. De modo que -11 y -3, -3 y +5, -8 y 0 serán valores admitidos que permitirán la ejecución del mismo número de operaciones.

Veamos ahora una palabra muy útil que se relaciona mucho con DO-LOOP. Se trata de [I] (representa la primera letra de Index) que realiza, nada más ser ejecutada, una copia del índice y lo sitúa en el stack. Puede ser incluida en cualquier punto que se desee entre DO y LOOP. El valor salvado de esta manera podemos imprimirlo, duplicarlo, añadirlo al segundo número en stack o podrá ser utilizado para cualquier otro fin como si de un parámetro cualquiera se tratase. Su forma general es:

valor final+1 valor de partida DO ... I ... LOOP

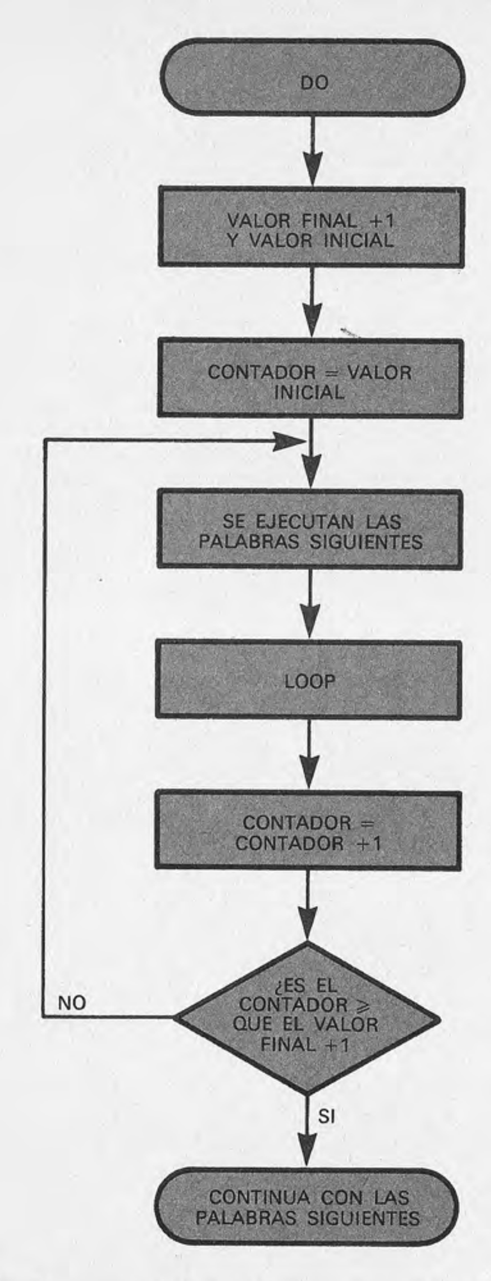


Figura 7.—Diagrama de flujo de una secuencia de DO-LOOP.

Incrementos no unitarios en los bucles

El FORTH permite también un nuevo tipo de bucle que admite incrementos (o decrementos, bastará con utilizar un valor negativo) distintos de 1. Su forma es

valor final+1 valor inicial DO ..(palabras).. n+LOOP

donde "n" es un valor cualquiera, positivo o negativo. Este bucle toma, como siempre, los dos valores que se hallan en la cima del TOS como límites del ciclo e incrementa el contador en un valor "n". Naturalmente, si "n" es positivo, el valor final será más grande que el inicial, de lo contrario será al revés (el índice tendrá que asumir, claro está, un valor inferior o igual al final).

Ponemos a continuación un ejemplo de utilización de esta palabra: queremos definir una palabra que calcule el factorial de un número. Incluimos entonces en el diccionario la palabra [!!] (no puede utilizarse el signo del factorial, !, ya que en FORTH sirve para otra cosa). Tendremos:

```
: !!                     (define la palabra del factorial recorda-
                         mos que el factorial de un número "n" es
                         el producto de los números consecutivos
                         entre 1 y n)
DUP                      (conserva una copia del número)
1+                       (incrementa el valor final)
1 DUP                    (valor inicial)
1+                       (lleva el valor inicial a 2 para evitar la pri-
                         mera multiplicación)
ROT                      (lleva el valor final al TOS)
DO
  I                      (escribe en el TOS el valor en curso del
                         bucle)
  *                      (calcula el primer producto del factorial)
LOOP                     (fin de bucle)
SWAP                     (intercambia valores inicial y final)
."el factorial de" .. "es".
CR
;                        (fin definición)
```

La definición posee una vez más abundantes comentarios. Esta es la única forma de que un programa resulte claro, pues de otra manera no sería siempre comprensible en su totalidad. Si analizamos el stack (Fig. 8) la definición resultará aún más clara. Hacemos sólo mención a los límites impuestos por los números de simple precisión, aunque, naturalmente, en caso de necesidad la



Figura 8.—Análisis del stack durante la ejecución de la palabra "!!". n=Número cuyo factorial se quiere calcular.

definición podría ser redactada utilizando números dobles y palabras específicas para ello.

Igual que en otros lenguajes, unos bucles pueden estar incluidos en otros; por ejemplo, cuando queremos inicializar o asignar valores a una tabla bi o multidimensional. Tales bucles se dice que están anidados y no presentan ninguna dificultad especial.

Prácticamente ya hemos terminado de hablar de la estructura DO-LOOP; sólo nos queda puntualizar algo sobre una cuestión que tal vez se habrá planteado ya algún lector. Veamos un ejemplo que será siempre más eficaz que cualquier definición; durante el bucle, ¿dónde se conserva el valor en curso del ciclo? Podría parecer obvia la respuesta: "¡se encarga de ello el sistema operativo!" Pero el FORTH debe su éxito y rapidez, precisamente, a la estructura tan ordenada y "transparente" para el usuario, quien puede conocer todo lo que ocurre en el sistema, al cual puede acceder en cualquier punto.

Todo sistema FORTH posee un segundo stack, el Return Stack, igual en todo al anterior, con la diferencia de que a éste el usuario no puede acceder directamente. Está organizado de la misma manera (estructura LIFO, direcciones decrecientes de memoria ocupadas, etc.) y el programa en sí podrá utilizarlo para algunas o, mejor dicho, muchas de sus funciones.

Por ejemplo, en el DO-LOOP el FORTH utiliza el Return Stack para llevar la cuenta de los valores del bucle, tanto de los intervalos como del nivel final. Al final de cada bucle el valor situado en el TOS del Return Stack es comparado con el valor final, presente en el segundo puesto, y si no es mayor o igual se incrementa con el valor del intervalo, presente en tercer lugar (algunos RS tienen intercambiado el puesto del segundo y tercer valor). Una vez que se verifican las condiciones de salida, los números son extraídos del RS, quedando así éste limpio para operaciones sucesivas.

El RS sirve para otras muchas funciones del sistema. Por ejemplo, la palabra [*/] (multiply then divide, multiplica y luego divide) conserva el resultado intermedio de doble precisión, precisamente en el RS. Una vez más la palabra [I] no hace otra cosa sino efectuar un DUP del RS en el TOS del Data Stack (el Stack del que habíamos hablado hasta ahora).

A pesar de que, como hemos dicho, el R5 está reservado en exclusiva para las necesidades del sistema operativo (algunos RS originariamente eran del todo inaccesibles), existen algunas palabras que actúan sobre él. Sin embargo, el RS es mucho más delicado y difícil de manejar que el DS, así que hay que poner mucho cuidado y atención al utilizar dichas palabras. Estas son:

Palabra	Efecto	Explicación
>R	n - - -	transfiere "n" desde el Data Stack al Return Stack;
R>	- - - n	transfiere "n" desde el Return Stack al Data Stack;
R@	- - - n	copia el número que está en el TOS del Return Stack en el Data Stack;
I	- - - n	Como R@; ya vimos su utilización al hablar del bucle;
I'	- - - n	copia el segundo número del Return Stack en el TOS del Data Stack;
J	- - - n	copia el tercer número del Return Stack en el Data Stack.

Como puede ver hemos incluido también la palabra [I], que ya conocíamos y que, en realidad, corresponde a la secuencia

R> DUP >R

Naturalmente la palabra [I] podrá ser utilizada fuera de un bucle también.

En resumen, el Return Stack puede servir como una "tercera mano" que conserva temporalmente datos para un próximo uso. A pesar de eso es aconsejable habituarse a emplearlo lo menos posible y, de todas formas, habrá que extraer los datos en él contenidos antes de terminar las operaciones de una definición (es decir, antes de la palabra [;]). Esto se debe a que la palabra [;] recupera en el TOS del RS un apuntador de memoria, así que no es posible utilizar el RS para transferir parámetros de una palabra a otra.

CAPITULO X

BUCLES CONDICIONALES

Tomando decisiones



e dan casos en los que el ordenador tendrá que vérselas frente a una elección condicional, es decir, que deberá tomar una decisión. En este capítulo trataremos precisamente de las instrucciones que permiten al ordenador tomar las decisiones que nosotros consideremos oportunas. En FORTH tales estructuras son, en muchos aspectos, parecidas a las de otros lenguajes; la diferencia estriba en que en el FORTH tal decisión se toma en base al valor verdadero-falso de un indicador situado en el TOS.

Este indicador, o bandera (flag, en inglés), se considera verdadero si su valor es (o, como suele decirse, está puesto a) 1 y falso cuando está puesto a 0. Una serie de palabras, llamadas de comparación, se encargan de efectuar las comparaciones necesarias. Vamos a mencionarlas a continuación con una breve explicación de su utilización y valor:

El significado de estas palabras se comprende fácilmente de modo que no requieren ulteriores explicaciones. Como es lógico, será necesario que estén en el stack los valores que hay que comparar. Por ejemplo:

```
3 5 < . CR
```

dará

```
1
OK
```

Palabra	Efecto	Explicación
0<	n - - - flag	pone el flag a 1 si n<0
0=	n - - - flag	igual si n=0
0>	n - - - flag	igual si n>0
<	n1 n2 - - - flag	pone el flag a 1 si n1<n2
=	n1 n2 - - - flag	igual si n1=n2
>	n1 n2 - - - flag	igual si n1>n2
<>	n1 n2 - - - flag	igual si n1 es distinto de n2
U<	n1 n2 - - - flag	como < pero con números sin signo
D0=	d - - - flag	lo mismo con números dobles
D<	d1 d2 - - - flag	lo mismo con números dobles
D=	d1 d2 - - - flag	lo mismo con números dobles
DU<	d1 d2 - - - flag	lo mismo con números dobles
NOT	flag 1 - - - flag 2	invierte el valor de la bandera
?DUP	n - - - n (n)	duplica "n" sólo si es distinto de 0.

Pero entonces los valores 3 y 5 desaparecerán del stack. Si se quieren conservar habrá que hacer:

```
3 5 OVER OVER < . CR
```

Dos tipos de bucles condicionales: Begin-Until y Begin-While-Repeat

El ciclo BEGIN-UNTIL es muy similar al DO-LOOP, pero el número de repeticiones que hay que efectuar no está especificado. En efecto, una operación se repetirá un número indefinido de veces hasta que (UNTIL) se verifiquen ciertas condiciones determinadas. Para ello tendremos que usar las operaciones de comparación que hemos definido anteriormente.

El bucle BEGIN-WHILE-REPEAT realiza una secuencia mientras (WHILE) que se cumpla una condición. La expresión de estos dos tipos sería:

```
BEGIN ... flag UNTIL
BEGIN ... flag WHILE ... REPEAT
```

donde los puntos suspensivos representan unas operaciones y flag es la bandera verdadero-falso presente en el TOS que ya mencionamos.

Existe una diferencia entre ambas formas: la primera repite las operaciones entre BEGIN y UNTIL hasta que el indicador se ponga a 1; la segunda forma repite las operaciones entre BEGIN, WHILE y REPEAT hasta que el indicador adopte el valor 0.

Pero la diferencia sustancial estriba en que la forma BEGIN-UNTIL es efectuada (recorrida) al menos una vez, ya que el test de comparación se hará al final del ciclo. La forma BEGIN-WHILE-REPEAT permite saltarse las operaciones entre WHILE y REPEAT en el caso de que el indicador tenga el valor 0.

La forma BEGIN-UNTIL resulta muy útil para el cálculo de las funciones convergentes o de valores que se aproximan indefinidamente a un valor final por medio de una secuencia repetitiva. El ejemplo más clásico es el cálculo de la raíz cuadrada de un número. Este puede efectuarse en un ordenador con el método de Newton, utilizado prácticamente en todos los lenguajes menos en BASIC, FORTRAN o Pascal, donde tal función está ya definida (SQR, SQRT o como mejor le pareció a quien escribió el lenguaje). En FORTH, en cambio, lo corriente es que no la tenga (siguiendo la filosofía de "hazlo por ti mismo") y he aquí que se nos presenta una estupenda ocasión para echarle una ojeada a un sistema sencillo y genial para calcular con la aproximación que se desee esa función, utilizando sólo las cuatro operaciones de una manera fácil.

El método de Newton para calcular una raíz cuadrada es la cosa más simple de este mundo: establece que si A es un valor aproximado de la raíz cuadrada de un número "n",

A1=(n/A+A)/2

es una aproximación mejor. Es decir, que cada vez que se aplique esta fórmula a un valor ya calculado, el resultado obtenido tendrá un valor más próximo al valor final de la raíz.

Vamos a poner un ejemplo. Queremos calcular la raíz cuadrada de 4225. El primer valor aproximado que utilizaremos será 2 (3, 1, 50 o cualquier otro, pues valdrá cualquiera). Tendremos:

```
A1=(4225/2+2)    =1057
A1=(4225/1057+2)=530
A1=(4225/530+2) =268
A1=(4225/268+2) =141
A1=(4225/141+2) =85
A1=(4225/85+2)  =67
A1=(4225/67+2)  =65
A1=(4225/65+2)  =65
```

con la gráfica de la función como aparece en la figura 9.

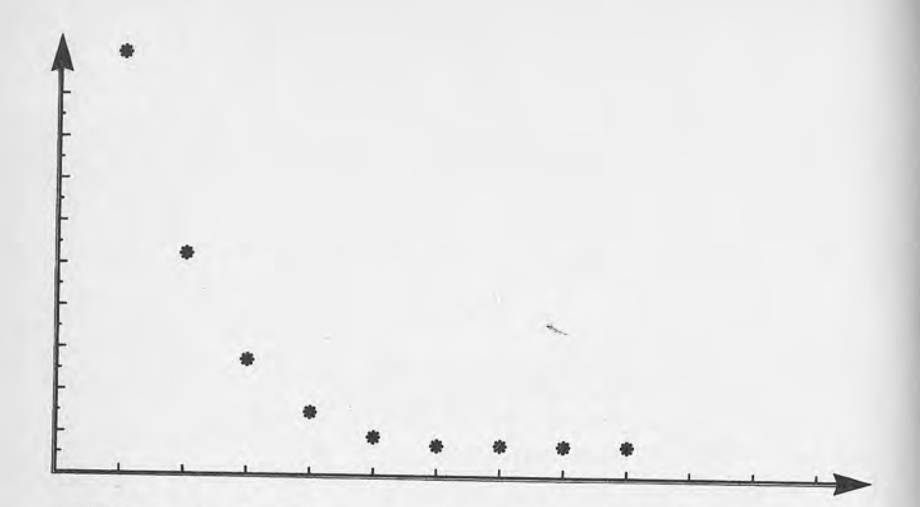


Figura 9.—Sucesión convergente de los valores de la raíz calculados con el método de Newton.

Vamos a implementar la palabra SQR para aquellos FORTH que no poseen raíz cuadrada, con la siguiente secuencia:

```
: SQR       (calcula la raíz cuadrada del número en
            TOS utilizando el método de Newton).
1           (propone 1 como primer valor aproxima-
            do; cualquier otro valor será válido)
BEGIN
  OVER      (estado del stack: n 1 n)
  0 3 PICK  (copia el tercer número en el TOS; esta-
            do del stack: 1 0 n L n)
  U/MOD     (estado del stack: cociente resto 1 n; re-
            cordamos que U/MOD divide un número
            doble por uno sencillo)
  SWAP      (estado del stack: resto cociente 1 n)
  DROP      (cociente 1 n)
  SWAP      (1 cociente n)
  OVER      (cociente 1 cociente n)
  =         (efectúa el test de comprobación para
            comprobar si el resultado es igual al va-
            lor aproximado precedente, en el primer
            ciclo igual a 1)
UNTIL       (empieza de nuevo desde el principio
            con el cociente que representa el nuevo
            valor aproximado, y con "n"intacto).
```



```
SWAP        ."la raíz cuadrada de"." es ".
;           (fin definición)
```

Una vez más los comentarios son abundantes (pero nunca superfluos en FORTH).

Algunos sistemas poseen otra forma de bucle del tipo

```
BEGIN ... AGAIN
```

que representa el verdadero bucle incondicionado, de modo que el sistema va continuamente adelante y atrás sin parar. Esto pone, en la práctica, al sistema en un "círculo vicioso" en espera, por ejemplo, de una interrupción o de un aviso por parte de un dispositivo de entrada/salida. Para simular esta forma en cualquier sistema basta con hacer BEGIN ... 0 UNTIL.

Las estructuras IF-THEN e IF-ELSE-THEN

También estas son estructuras condicionales. Con la primera es posible saltar o no una operación; con la segunda, escoger entre dos operaciones

```
flag IF ... (flag=1) ... THEN
flag IF ... (flag=1) ... ELSE ... (flag=0) ... THEN
```

Como puede verse, aparece una vez más la mano potente de la RPN que impone la presencia del valor (0 ó 1 del indicador) antes del operador (la palabra IF). En ambos casos IF hace un test (¿verdadero o falso?) de la bandera situada en el TOS.

Si flag=1 se efectúa la operación entre IF y THEN, en la primera estructra, y en la segunda estructura, la que está entre IF y ELSE. En caso contrario se saltará la operación en la primera, mientras que en la segunda se ejecutará la secuencia situada entre ELSE y THEN.

Imaginemos, por ejemplo, que queremos programar una máquina que pesa los equipajes en un aeropuerto para que los agrupe en base a su peso según que éste sea superior o inferior a 15 kilos. Cada vez que pongamos una maleta en la báscula se efectuará la secuencia:

```
15>
IF
 ."El peso supera los 15 Kg"
 CR
ELSE
```

."Peso inferior o igual a 15 Kg"
THEN

Los tests condicionales pueden estar también anidados uno dentro del otro. Supongamos que queremos clasificar las maletas según su peso de la siguiente forma: hasta 10 kilos, entre 10 y 20 kilos, y más de 20 kilos. Tendremos entonces:

```
: COMPARACION          (permite comparar un número o incluirlo
                       en una de las siguientes clases: n<10;
                       10<n<20; n>20)
DUP                    (duplica el TOS para utilizarlo eventual-
                       mente después de ELSE)
10<                    (efectúa el primer test)
CR
IF
 ."equipaje que pesa
 menos de 10 Kg"
 DROP                  (borra el valor duplicado y sigue hasta el
                       final)
ELSE
 20>
IF
 ."equipaje que pesa
 más de 20 Kg"
ELSE
 ."equipaje que pesa
 entre 10 y 20 Kg"
THEN                   (fin del bucle interior)
THEN                   (fin definición)
;
```

Hay, por último, dos palabras que permiten salir anticipadamente de un bucle: se trata de LEAVE y EXIT. La primera hace posible la salida del bucle al final del ciclo y asigna al límite el valor en curso en el contador. EXIT es aún más drástica, pues concluye inmediatamente una definición, es decir, funciona como una palabra [;] englobada dentro de la misma definición. Esta segunda definición funciona y debe ser utilizada fuera de un ciclo DO-LOOP.

Evidentemente, ambas palabras deberán estar controladas con un IF condicional: según el resultado de éste se efectuará (o no) la salida forzada del bucle.

CAPITULO XI

VARIABLES, CONSTANTES Y TABLAS



Ya hemos visto que el FORTH utiliza el stack como un área de estacionamiento temporal de datos para hacer cálculos, comparaciones o también transferir informaciones de una palabra a otra. Si, en cambio, lo que queremos es conservar números y valores de forma más permanente o bien tenemos que recurrir a esos números a menudo o si tales cifras han de ser puestas al día cada cierto tiempo, habrá que tener a nuestra disposición un medio más estable y fijo que nos permita poder hacer referencias rápidas y sencillas.

Las variables

En FORTH una variable se define simplemente al utilizar la palabra VARIABLE seguida del nombre. Por ejemplo, la secuencia

```
VARIABLE GONZALO
```

efectúa dos operaciones: incluye el nombre GONZALO en el diccionario y reserva dos bytes de memoria para alojar el valor que tal variable representará. Una vez definida dicha variable existe aunque no esté inicializada (es decir, no contenga nada). Para asignar un valor a la variable GONZALO, por ejemplo 320, hay que utilizar la palabra [!] (store) de la siguiente manera:

```
320 GONZALO !
```

El FIG-FORTH, generalmente más pobre y algo rígido, es en este caso más práctico y eficaz, pues en él ambas operaciones se pueden combinar en la secuencia:

320 VARIABLE GONZALO

El hecho de cambiar el valor de una variable resulta igual de rápido: será suficiente con volver a efectuar la operación. Así

420 GONZALO !

es lo único que habrá que teclear.

Para conocer el valor de una variable se necesita la palabra @(recuperar), de forma que al escribir

GONZALO @ .

obtendremos por respuesta

420 OK

Existe, por otra parte, la palabra [?] que podemos definir de la siguiente manera:

: ? @ . ;

entonces la secuencia

GONZALO ?

dará por respuesta el mismo resultado que antes.

De todas formas, cualquier FORTH tiene un pequeño grupo de variables predefinidas. Una de ellas es [BASE], que permite establecer el sistema de numeración deseado. Generalmente hay también algunas variables predefinidas de numeración de uso corriente, como HEX (hexadecimal), DEC o DECIMAL, OCT (octal) y también BIN (binario). Pero siempre podrán ser definidos distintos sistemas de numeración con secuencias del tipo:

```
: BASE5    5 BASE ! ;
: BASE9    9 BASE ! ;
```

o como mejor les parezca. Aunque la utilidad de estas bases "raras" (distintas de 2, 4, 8, 10 ó 16) no es grande.

En resumen: el uso de VARIABLE es parecido a la operación de introducción de una nueva palabra en el diccionario, con la excepción de que esto se efectúa con la palabra VARIABLE en lugar de con la secuencia [:] y [;]. En este caso, naturalmente, no será necesario especificar cuál es la función de esa palabra.

En la práctica lo que ocurre al escribir

50 GONZALO !

podemos esquematizarlo de la siguiente manera:

— 50 se sitúa en el stack;
— el intérprete busca en el diccionario GONZALO;
— cuando lo ha encontrado ejecuta inmediatamente una palabra específica (EXECUTE) que busca y copia en el stack la dirección de la palabra GONZALO;
— la palabra [!] toma del stack la dirección y el valor 50, va a la posición hallada por EXECUTE y coloca allí el valor 50, borrando cualquier otro que existiera anteriormente.

Lo mismo se podría decir con las pertinentes modificaciones, para [@] o [?]. Por ejemplo, con la frase

GONZALO @

ocurrirá lo siguiente:

— el intérprete busca en el diccionario GONZALO;
— ejecuta la palabra EXECUTE;
— una vez recibida la dirección va a esa posición y toma una copia del valor que hay allí.

Debería haber quedado claro el proceso, pero aunque así no fuera no es demasiado importante, ya que al menos al comienzo es suficiente con saber utilizar bien la palabra VARIABLE, cosa que, por otra parte, no requiere ningún esfuerzo particular.

Hay una palabra especial para las variables que resulta muy útil: se trata de [+!] (sumar-guardar), que permite sumar el número presente en el stack al valor de una variable. Por ejemplo, si la variable JUAN

VARIABLE JUAN

contiene el valor 30

30 JUAN !

la ejecución de

20 JUAN +!

y

JUAN @ (o bien JUAN ?)

dará

50 OK

La palabra [+!] es especialmente útil a la hora de utilizar un contador, un totalizador. Como es lógico, también es posible hacerse uno mismo las restantes operaciones (en caso de que no las haya) de la siguiente manera:

```
: -! SWAP NEGATE SWAP +! ;
: *! DUP @ ROT * SWAP ! ;
: /! DUP @ ROT / SWAP ! ;
```

que sirven para restar, multiplicar o dividir por un número la variable.

Se pueden también utilizar las variables para operar entre sí. De esta forma:

VARIABLE FELIPE GONZALO JUAN + SWAP !

creará la variable FELIPE, que contiene la suma de los valores que se hallan en las dos variables anteriormente definidas.

Constantes

Las variables que hemos descrito están destinadas a contener valores que pueden, como hemos visto, cambiar. Las constantes, en cambio, son estructuras del lenguaje cuyos valores, generalmente, no cambiarán. Un programa de facturación contendrá, por ejemplo, la constante del IVA (de valor igual a 12) destinada, al menos en principio, a no cambiar durante un largo período de tiempo.

La manera de definir una constante es muy parecida a la de una variable; bastará con escribir

12 CONSTANT IVA

para que la constante IVA tome el valor deseado.

A primera vista podría parecer que todo esto es una inútil complicación o una copia de VARIABLE, pero no es así. Al decir IVA, el sistema operativo se referirá a un valor y no a una dirección, como en el caso de las variables. Las operaciones de búsqueda (@), por tanto, no tendrán sentido alguno, ya que no está implicada la función de EXECUTE y no hay indicaciones hacia direcciones de memoria.

La verdadera utilidad de las constantes reside en que conservan datos a los que acceder inmediatamente y que están, digámoslo así, cristalizados o congelados en el sistema operativo. El acceso directo a un valor que se produce con las constantes es mucho más veloz y eficaz que al buscar una dirección desde donde se señalará luego un valor (aunque al usuario tal vez no le parezca importante la diferencia que existe); todo ello comportará una mayor eficiencia y, por tanto, más velocidad operativa. No hay que olvidar que para un programador en FORTH lo importante no es que el ordenador haga cualquier cosa o a cualquier precio, sino que la realice de la mejor manera posible; está claro que será mucho más conveniente utilizar constantes, aunque pueda parecer a veces una complicación inútil.

Queremos resaltar un detalle: si escribimos

30 CONSTANT LUIS

equivale, o al menos produce, el mismo efecto que

: LUIS 30 ;

pero esta definición ocupa más memoria y tiempo (alrededor del 40 por 100) en la fase de ejecución.

Lo corriente es que una constante, una vez inicializada, conserve su valor indefinidamente, pero podría ocurrir que en un momento dado hubiera que cambiar el valor ya existente. Para ello habrá que ejecutar la siguiente operación:

n1 ' nombre de la constante !

en la práctica:

63 ' LUIS !

sustituirá el valor 30 por el de 63.

Seguro que ahora más de un lector se estará preguntando: "si las constantes son mucho más eficientes, ¿por qué no se utilizan siempre en lugar de las variables? Cuando sea necesario bastará con cambiar el valor de la constante con la secuencia que ha sido

descrita". Pero este razonamiento no es válido, ya que dicha secuencia ['nombre!] es mucho más lenta que cualquier operación con variables.

Un último consejo: debido a la gran exigencia de autodocumentación de los programas en FORTH y para evitar confusiones es conveniente utilizar nombres para las variables y, aún más, para las constantes que sean muy significativos. Así, por ejemplo, la constante de Planck es mejor definirla como PLANCK en lugar de PK. Además, cada sistema acepta identificadores de distinta longitud, así que será mejor hacer pruebas (o leerse atentamente el manual) para evitar que PARALELEPIPEDO y PARALELOGRAMO puedan significar lo mismo.

Es posible definir variables y constantes con números de doble precisión. Las palabras que habrá que utilizar son 2VARIABLE y 2CONSTANT además de [2!] y [2@]. Las reglas serán las mismas cambiando sólo la modalidad de instrucción del número doble; como ya dijimos, para indicar que se trata de un número de doble precisión habrá que poner un punto en el número: así 10.0000, 10000.0 y 1.00000 son, cualquiera de ellos, válidos con un valor de 100000.

Las tablas

Son un conjunto de datos homogéneos dispuestos en orden. De tal forma será una tabla (array, en inglés) la lista de la compra de un ama de casa, el extracto de los movimientos de una cuenta bancaria, la tabla de pesos atómicos de los elementos e incluso la lista de los santos en las páginas del calendario. La utilidad que poseen las tablas es que en lugar de referirse al elemento es posible "jugar" con el puesto que éste ocupa dentro de ella.

En la práctica es lo mismo que ocurre al marcar un número de teléfono. Las tablas pueden estar organizadas como vectores (unidimensionales, es decir números organizados sólo en líneas) y como matrices, donde cada elemento se localizará por el número de línea y columna que ocupa. En tal caso se llaman tablas bidimensionales, de dimensiones a×b, siendo éste el caso de los cuadros de un crucigrama o de la batalla naval. Hay también matrices tridimensionales a×b×c y pluridimensionales que son, conceptualmente, iguales en cuanto al sistema de funcionamiento y localización se refiere.

Vamos a dar por un momento un paso hacia atrás; cuando definimos una variable con la secuencia

VARIABLE ALBERTO

el sistema operativo reserva un espacio en memoria tal y como se muestra en la figura 10a. La palabra ALLOT justo después de la propia definición de la variable en la forma

VARIABLE NOMBRE n ALLOT

permite reservar otros "n" bytes de espacio al lado de los anteriores (Fig. 10b); es entonces posible afirmar que en las variables dimensionadas de esta manera se podrán conservar dos números sencillos o un número doble, al tener a disposición 4 bytes. Por tanto

VARIABLE JAVIER 8 ALLOT

permitirá reservar espacio para cinco números sencillos; más genéricamente, ALLOT dimensiona la variable de forma que pueda contener (n+2)/2 números enteros y (n+2) MOD 2 bytes restantes. Es lógico que si estos últimos sobran serán de poca utilidad, así que es conveniente que "n" no sea nunca impar (al menos por el momento) y que, al trabajar en doble precisión, n+2 sea múltiplo de 4.

Ahora las cosas empiezan a complicarse. Imaginemos que tenemos un vector destinado a contener "n" gastos mensuales; empezamos con

VARIABLE GASTOS 22 ALLOT

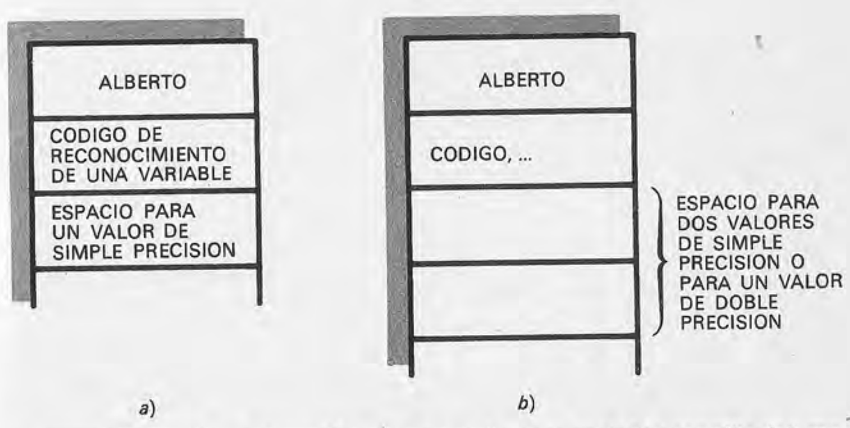


Figura 10.—Asignación de memoria: para variable sencilla (a); mediante el uso de [ALLOT] (b).

Tendremos así una tabla monodimensional de doce puestos de dos bytes cada uno reservados para conservar los valores de los gastos mensuales. Ahora bien, ¿cómo se colocan los valores del mes de abril o de diciembre en la casilla correspondiente? Para ello tenemos las constantes. Vamos a definirlas

```
0  CONSTANT  ENERO
2  CONSTANT  FEBRERO
4  CONSTANT  MARZO
```

... y así hasta

```
20  CONSTANT  NOVIEMBRE
22  CONSTANT  DICIEMBRE
```

Recordamos que al escribir el nombre de la variable estamos reclamando la dirección (y no el valor); así, si queremos meter 300 en el mes de marzo escribiremos

```
300  GASTOS  MARZO  +  !
```

y el valor 300 estará incluido en el lugar correcto.

Naturalmente, también podrán utilizarse las otras palabras específicas para las variables; así, si quisiéramos añadir el valor 32 al 300 que ya está almacenado, bastará con utilizar [+!] en lugar de [!] (por otra parte, ya se habrán dado cuenta de que la palabra [+!] corresponde a la secuencia "n nombrevariable @+nombrevariable !").

Inicializar una tabla puede significar también asignar a las distintas posiciones distintos valores. Supongamos que un concesionario quiere construir una tabla que contenga todos los precios de las piezas de recambio. La formará entonces de la siguiente manera:

```
VARIABLE  RECAMBIOS  298  ALLOT
```

Esta podrá contener los precios de 150 piezas diferentes. Luego construirá las constantes de correspondencia: nombre de la pieza menos incremento de dirección. Y viene ahora la labor, tan pesada, de meter uno por uno los precios con la técnica

```
1200  RECAMBIOS   !     (precio del volante)
 700  RECAMBIOS  2+!    (precio del piloto)
 250  RECAMBIOS  4+!    (precio de la bujía)
...
```


Podrán imaginarse lo aburrido que resulta. Por suerte existe una palabra [,] que puede considerarse una combinación de ALLOT y [!]. Toda la operación será entonces:

```
VARIABLE  RECAMBIOS
1200      RECAMBIOS
700, 250,... (y todos los demás)
```

¡Nótese la diferencia! No se requiere indicar inicialmente el valor de ALLOT, ya que la palabra [,] dimensiona dinámicamente la variable RECAMBIOS, es decir, la incrementa cada vez que aparece una magnitud de dos bytes y se encarga automáticamente de almacenar el valor.

Si fuera necesario crear en un programa varias tablas podemos evitarnos tener que efectuar la definición y asignación de cada una. Con todo lo que hemos visto hasta ahora y con dos nuevas palabras [CREATE] y [DOES>] se puede automatizar todo.

La primera, CREATE, incluye un nombre en el diccionario, pero sin reservar espacio para él. Así la secuencia

```
CREATE  RECAMBIOS
```

pondrá el nombre RECAMBIOS en el diccionario, aunque no le reserve ningún espacio (funciona sustancialmente como VARIABLE, pero sin los dos bytes para almacenar datos).

Si combinamos CREATE con DOES> podremos crear una estructura elástica capaz de resolver, de una manera muy práctica, varios problemas. Tendrá la siguiente forma:

```
: nombre  CREATE  ...  DOES>  ...  ;
```

donde la secuencia entre CREATE y DOES> especifica qué ocurre cuando la definición está compilada y las palabras correspondidas entre DOES> y [;] especifican qué hacer cuando un objeto de la clase construida por CREATE es reclamado o ejecutado.

Vamos a poner un ejemplo: supongamos que queremos construir una palabra, ARRAY, que crea una tabla en el diccionario de, por ejemplo, 12 puestos. Seguiremos paso a paso su funcionamiento. Escribimos siguiendo el ejemplo anterior

```
12  ARRAY  RECAMBIOS
```

CREATE incluye el nombre RECAMBIOS en el diccionario, el valor 12 se duplica y ALLOT reserva 24 bytes (12 puestos) en memoria. Recuérdese que todo lo que se halla contenido entre CREATE y DOES> especifica qué ocurre cuando se compila la

definición, así que en el momento de la compilación se reservan 24 bytes para RECAMBIOS.

La segunda parte permite obtener la dirección de un elemento multiplicando por dos su número de orden y añadiendo el valor a la dirección base. Así

2300 6 RECAMBIOS !

efectuará las operaciones siguientes:

— RECAMBIOS deja en TOS la dirección de base;
— según la definición de array, 6 pasa al TOS y la dirección al segundo puesto (SWAP);
— el número de orden 6 se multiplica por 2, para obtener la posición relativa en memoria;
— se evalúa la verdadera posición del elemento, sumando dirección de base relativa;
— 2300 se almacena en la dirección.

De la misma manera tendremos

6 RECAMBIOS ? 2300 OK (valor presente en el octavo elemento)
500 6 RECAMBIOS +! OK (500+2300 en el octavo elemento)
6 RECAMBIOS ? 2800 OK (valor puesto al día en el octavo elemento)

CAPITULO XII

DIALOGANDO CON EL FORTH

Hasta ahora hemos visto cómo usuario y ordenador desempeñaban el papel, digámoslo así, de escritor y actor: el primero preparaba el guión (el programa) e informaba al actor (el ordenador) acerca de lo que tenía que hacer; el segundo leía su papel (el programa) y lo interpretaba sin pedir más explicaciones.

Pero en la vida real se da con frecuencia el tener que ir metiendo en el programa datos que no han podido ser determinados de antemano. Este es el caso, por ejemplo, de la mayor parte de las aplicaciones comerciales, donde hay una cantidad tremenda de datos muy variables en el tiempo. Piense en un programa de facturación y se dará cuenta que resulta mucho más práctico que el programa en curso solicite por sí mismo algunos datos antes que meterlos todos y poner en marcha el programa cada vez.

Lo mismo ocurre con programas matemáticos o de ingeniería, en los que el volumen de datos es tal que sería poco práctico introducirlos en el stack desde un principio (¿se imagina las dificultades a la hora de manejarlos?) arriesgándose incluso a saturar el stack que, desde luego, no es un pozo sin fondo... Y, sin ir más lejos, hasta en un juego es necesario que interactúe durante la ejecución con el jugador para que este último pueda decidir el movimiento oportuno en su enésima batalla contra los marcianitos.

Por otro lado, con frecuencia un programa debe proporcionar resultados parciales que han de ser enviados a una impresora o a la pantalla. En realidad ya conocemos algunas de estas técnicas, como las palabras [.], [CR], [." "], pero no son desde luego suficientes para resolver todos los problemas del diálogo hombre-máquina.

Operaciones de I/O

Las operaciones que necesitamos se llaman de I/O (input/output) y sirven para recibir órdenes del teclado o transmitir resultados o peticiones a una impresora o a la pantalla. Estas informaciones se transmiten generalmente por medio de un código específico de 8 bits llamado ASCII (American Standard Code for Information Interchange) que es en la práctica algo así como el alfabeto estándar de los ordenadores.

La principal operación de entrada (input) es la que espera, por parte del teclado o de un instrumento de medida, recibir una señal formada por un solo código. Esto quiere decir, de forma más sencilla, que el ordenador aguarda a que el operador apriete una tecla o a que el aparato al que está conectado le envíe un dato en forma de código ASCII.

La palabra [KEY] se encarga de ello: ésta pone al ordenador en una situación de espera hasta que sea pulsada una tecla. El valor del código respectivo se sitúa entonces en el TOS. Por tanto si escribe

```
KEY
```

y aprieta a continuación la tecla "P" obtendrá OK sin otro mensaje. Pero si pulsa entonces la palabra [.] el ordenador responderá

```
80 OK
```

que representa, precisamente, el valor ASCII del carácter "P".

Lo cierto es que 80, de por sí, no nos dice mucho a menos que tengamos una memoria fabulosa. Existe entonces otra palabra [EMIT], que convierte el valor numérico en el carácter ASCII correspondiente, es decir, en la práctica ejecuta la operación inversa de [KEY] y equivale al CHR$(n) del BASIC. Así

```
: AA KEY EMIT ;
```

dará, cuando apretemos la tecla "P"

```
P OK
```

Podemos mejorar aún más la definición con

```
: AAA KEY DUP ."Ha sido pulsada la tecla" EMIT CR;
```

que dará como salida (output))


```
Ha sido pulsada la tecla "P"
```

El valor 80 se conservará en el TOS para usos posteriores, ya que es bastante improbable que nadie se siente frente a un ordenador para apretar teclas y esperar a que el ordenador le diga cuáles ha pulsado...

Hay al menos otras tres palabras especialmente útiles en las operaciones de I/O. La primera, [BELL], envía al periférico el carácter ASCII 7 (bell=campana), que activa el timbre de la impresora o del monitor. [BL] en cambio, deja en el TOS el valor 32 (Blank que corresponde a la pulsación de la barra espaciadora). Por último [SPACE], que podemos definir como

```
: BL EMIT ;
```

transmite al periférico de salida un espacio (ASCII 32, Blank) sin incluir nada en el TOS. Como es natural, si utilizamos estas palabras para formar otras a nuestra medida, es decir para nuestro uso y beneficio, podremos crearnos una salida más adecuada.

Entrada/salida de cadenas

Lo corriente es esperar del teclado una secuencia de caracteres más que letras individuales. Con más precisión, se espera como entrada una cadena (string, en inglés) que representa números, partes de texto (frases, comunicados, respuestas, etc.) o cualquier combinación de ambas cosas. En honor a la verdad, el hecho de que sea una cadena tiene significado sólo para el usuario, ya que el sistema operativo la trata y considera como una serie de bytes consecutivos en los que cada uno representa el valor correspondiente a un carácter en la secuencia planteada.

La cadena puede estar formada de muchas maneras. El FORTH la conserva como una serie de caracteres (o mejor, como una serie de códigos que representan caracteres) precedida de un byte que indica su longitud. Como el valor máximo que puede representar un byte es de 255, ésta será la longitud máxima, en caracteres de una frase que pueda ser manejada por el sistema operativo sin necesidad de instrucciones adicionales (como, por ejemplo, de dimensionamiento).

La cadena, delimitada al comienzo por el contador de caracteres, posee al final el CARRIAGE RETURN (CR; CTRL-M, ASCII 13). El orden según el cual está situada en memoria es creciente. Muchos Forths no colocan al final de la línea el carácter ASCII 13. De esta manera la cadena se individuará al final por el primer carácter NUL (ASCII 0).

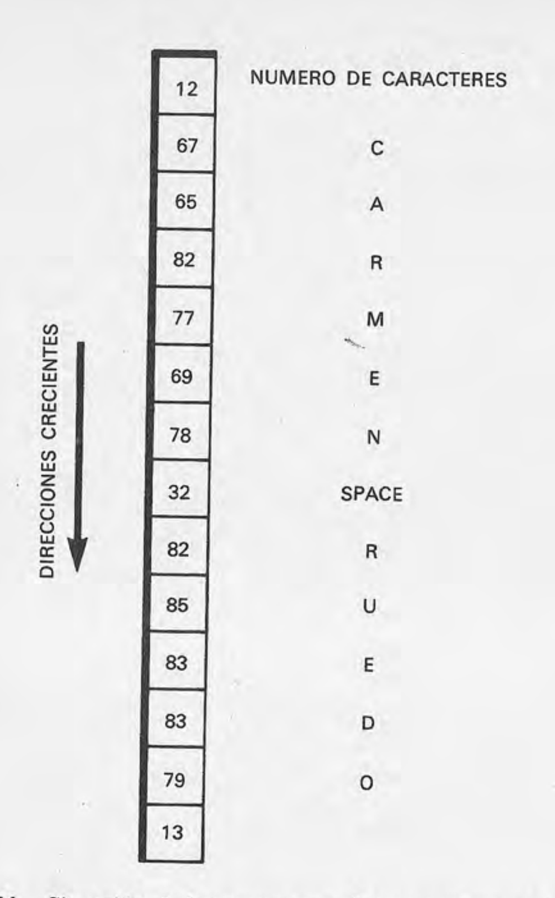


Figura 11.—Situación de los caracteres de una cadena en la memoria.

Decíamos que es posible dar por entrada una frase en lugar de un solo carácter. Hay para ello dos palabras: [EXPECT] y [TYPE]. Ambas extraen del stack dos parámetros: la dirección de partida y el número de caracteres según la fórmula general

dir n xxxxx - - -

donde xxxxx representa una de las dos palabras.

La primera, [EXPECT] acepta caracteres del teclado o de otro medio de entrada, hasta haber recibido las "n" letras o hasta que haya sido pulsada la tecla RETURN. Cada vez que se aprieta una tecla su valor se pone en memoria en una posición más alta que la del carácter anterior. Cuando la cadena está agotada el sistema

se encarga de añadir automáticamente uno o dos caracteres nulos (ASCII 0) al final.

Como [EXPECT] no tiene en cuenta la longitud de la cadena, sino sólo la máxima, no hay forma de conocer su longitud exacta. A menos que se quieran contar cada vez las letras con los dedos, será mejor construirse una palabra que se encargue de calcular dicha longitud como diferencia entre la dirección más alta no ocupada por el ASCII 0 y la de partida. Esto puede hacerse con un simple bucle que compruebe desde la dirección inicial uno por uno todos los caracteres hasta encontrar uno igual a 0 (o hasta hallar CR, si el sistema incluye esta palabra al final de la línea en la fase de almacenamiento en memoria).

La palabra [TYPE] hace lo contrario que [EXPECT], esto es:

dir n TYPE

efectúa la extracción, a partir de la dirección dir, de una cadena de n-1 caracteres.

Existe para apoyar a ésta la palabra [COUNT], que es especialmente útil, pues con ella no hace falta tener en cuenta la longitud de las cadenas almacenadas. En la práctica [COUNT] no es otra cosa sino una variable predefinida destinada a contener la dirección final de la cadena. El hecho de tener esto en cuenta resulta muy útil, por ejemplo, a la hora de extraer una subcadena (parte de la cadena principal). Así las formas genéricas

dir COUNT n - TYPE
dir n - COUNT TYPE

permite obtener, respectivamente, la primera y segunda parte de una cadena.

Más genéricamente es posible crear una palabra de extracción de subcadena según la fórmula

dir n1 n2 SUBCADENA - - -

en la cual podemos definir la palabra SUBCADENA como

: SUBCADENA ROT ROT + TYPE ;

que, por su sencillez, no requiere comentario alguno.

Hay una palabra específica para las cadenas: [PAD]. En efecto, existe un área de estacionamiento en memoria que tiene una amplitud mínima de 64 bytes (como se especifica en el estándar 79) y que puede utilizarse para conservar cadenas reservadas para una manipulación intermedia. Este área está ubicada dinámi-

camente, o sea, que puede ser desplazada en memoria cuando se necesite, a una distancia fija del área del diccionario. Esto determina precisamente su variabilidad de dirección en base a las dimensiones del diccionario y a las palabras añadidas posteriormente. Pero éste no es un problema, ya que el área del PAD (es decir, almohadilla, tampón) suele utilizarse dentro del ámbito de una misma palabra, así que no se producen variaciones relativas de la distancia diccionario-dirección del PAD.

La palabra [PAD] reclama la dirección de partida del área intermedia y la pone en el TOS. De esta manera hay 62 espacios disponibles para letras (64 menos un contador y menos un valor NULL que hay que situar al final). En FIG-FORTH, en cambio, el área del PAD es de 68 puestos, es decir 66 espacios para letras.

Hay otras dos palabras dedicadas al formateo. La primera, [SPACES], toma del TOS un número e imprime la misma cantidad de espacios. Resulta útil sobre todo para encolumnar y, unida a la palabra precedente y a la longitud de la cadena que hay que modificar, permite la preparación de una palabra análoga al PRINT USING del BASIC.

La segunda palabra es [-TRAILING] que elimina los espacios contenidos en la cadena superiores a 1. Se trata de una operación que actúa sólo sobre la salida: la cadena en memoria y el contador permanecen inalterados.

Del mismo modo que para las cadenas, hay también palabras destinadas a formatear números. [.R], [.UR], [.DR] son algunas de ellas y sirven para imprimir números sencillos sin signo o dobles alineados a la derecha. No vamos a describir estas palabras ya que varían mucho según las distintas versiones del lenguaje; es mejor que el usuario las estudie en el manual de instrucciones que le ha sido entregado.

Entrada/salida numérica

Nos hemos referido hasta el momento a la introducción de cadenas, pero, sobre todo en aplicaciones científicas, lo que posee una importancia fundamental es la entrada numérica.

Antes de seguir adelante, una precisión: cuando efectuamos una operación numérica utilizando un ordenador programable en FORTH, el sistema operativo trata de interpretar todo lo que recibe del teclado en el siguiente orden:

— comprueba que no se trate de una palabra;
— trata de interpretar la secuencia como valor numérico, transformándolo y lo pone en el TOS;
— trata de valorar una cadena;

— en caso de que todo esto no sea posible envía un mensaje de error.

Todo esto es cierto siempre que se utilice el ordenador como calculadora o cuando, antes de la llamada de la palabra, quiera meter datos en el stack. Pero en forma interactiva las entradas por teclado se consideran como una secuencia de caracteres alfanuméricos (en la práctica, como letras de una frase) como ya vimos.

Para que pueda interpretarse una secuencia como número hay dos palabras específicas: [NUMBER] y [CONVERT]. La primera toma del stack un número doble y una dirección y devuelve un número doble y una dirección. Podemos representarla de la siguiente manera:

```
n1  dir1  - - -  n2  dir2
```

donde n1 y n2 son números de doble precisión.

En realidad, n1 es un número sin significado que sirve sólo para reservar en el stack un espacio igual a 4 bytes. Dir1 es, en cambio, la dirección del byte que precede al primer carácter de la cadena. [CONVERT] no necesita este byte, puesto que no precisa conocer la longitud de la cadena.

Partiendo de la dirección dir+1 [CONVERT] empieza a traducir cada carácter ASCII a su valor numérico y acumula ese valor en n1. Como puede verse, el contenido asignado inicialmente a n1 no tiene importancia alguna y puede ser, incluso, igual a cero. El proceso sigue hasta que se detecte un carácter que no sea transformable en número. Entonces en el segundo puesto del stack se hallará n2 y se almacenará en el TOS la dirección del primer carácter no traducible (dir2).

Para hacer todo esto aún más sencillo podemos utilizar el PAD para almacenar la cadena de entrada. Pongamos un ejemplo. Supongamos que queremos simular una entrada numérica, es decir, que en un momento dado queremos suspender la ejecución de un programa para esperar un dato numérico que queremos recibir desde el teclado. Construimos la palabra INPUT:

: INPUT	(interrumpe el programa y espera la introducción de un dato desde el teclado o desde un periférico de control)
. " ? " CR	(imprime un carácter de petición)
PAD	(solicita el área del PAD)
11 EXPECT	(longitud máxima del número: 10 cifras)
0	(valor de n1 para el CONVERT)

0 PAD 1 -	(sitúa en el TOS la dirección de partida de la secuencia cadena-número)
CONVERT	(efectúa la conversión)
DROP	(borra la dirección final)
SWAP	(pone en el TOS la dirección de la variable)
!	(conserva en la variable el valor descifrado)
;	(fin definición)

Esta definición funciona como el INPUT en BASIC. La única diferencia estriba en que aquí la variable debe estar ya mencionada. Por ejemplo, tras escribir

VARIABLE A

la secuencia

A INPUT

funcionará de la misma manera que la línea BASIC

```
.............
400 INPUT A
.............
```

Con la única diferencia formal de que la sintaxis operando-operador estará impuesta, como siempre, por la notación polaca inversa. Pero tiene una gran ventaja: mientras que en BASIC la secuencia se utiliza sólo durante la ejecución, en FORTH esto es posible también interactivamente desde el teclado, lo que no es poco...

Naturalmente la definición de [INPUT] compilada de esta manera puede ser complementada con estructuras de control que verifiquen y soliciten confirmación del valor que ha sido escrito. Esto es posible introduciendo en la palabra una secuencia de control, por ejemplo después del [EXPECT], con IF condicionados efectuando el tipo de control que hay que ejercer.

La palabra [NUMBER] funciona de manera parecida a [EXPECT], pero acepta, en la cadena que hay que convertir a número, el signo "-", de forma que también permite la entrada de números negativos. Naturalmente, la presencia de dos palabras diferentes indica que existen diferencias de velocidad entre ambas, pues de otro modo no tendría sentido la presencia de [EXPECT].

CAPITULO XIII

LAS BASES DE NUMERACIÓN EN FORTH

A partir de ahora daremos por supuesto que el lector sabe utilizar las numeraciones binaria, octal y hexadecimal, en particular esta última, en la cual se representan las cifras superiores al 9 con letras de la A a la F. Quien no cumpla esta condición puede repasar los primeros libros de la B.B.I., donde se estudiaron más extensamente.

Al contrario que la mayoría de los lenguajes, que admiten como mucho 3 ó 4 sistemas de numeración (binario, decimal, hexadecimal y, a veces, octal), el FORTH admite nada menos que 69 distintos, en particular del 2 al 70. El valor del sistema de numeración en curso se halla en la variable [BASE].

Al encenderlo, el sistema inicializa la variable al valor 10 (BASE 0. dará 10 OK). Si queremos pasar a otro sistema bastará con efectuar una asignación de variable en la forma

n BASE !

que ya conocemos.

De tal forma, si tuviéramos que pasar al sistema hexadecimal tendríamos

16 BASE !

El hecho de que el ordenador no haga ninguna diferencia entre los sistemas se pone de relieve por la secuencia

: AAA	(muestra los resultados de una suma en dos sistemas de numeración distintos; hexadecimal y decimal)

```
16 BASE
+ DUP
."el total de los dos primeros números del stack es"
CR
."en hexadecimal: " .CR
A BASE !
."en decimal: " .CR
;
```

que da por respuesta la suma de los números que se hallan en el TOS en dos sistemas distintos de numeración.

Pero una vez más aparece algo que puede resultar extraño: la secuencia "A BASE !". Trataremos de llegar a su lógica con cierto orden; supongamos que estamos en hexadecimal y queremos pasar a decimal; tecleamos entonces:

```
10 BASE !
```

pero estaremos cometiendo un grave error, pues seguiríamos estando en hexadecimal. ¿Por qué? Porque 10 en base 16 vale, precisamente, 16. Como nos hallamos todavía en base 16 (representado por la cifra 10) para volver a base 10 habrá que indicar ese valor en la base que está aún en vigor (en base 16 , 10 vale precisamente A). ¿Queda claro?

De todas formas hay dos palabras específicas que sitúan el sistema en las bases numéricas más corrientes, no teniendo que complicarse la vida con estos cálculos. Estas palabras son [DECIMAL], que pone inmediatamente la variable [BASE] al valor decimal 10, y [HEX], que lo hace con el valor 16 (base hexadecimal). La gran ventaja de estas palabras es que con ellas no hay que tener en cuenta las bases iniciales de partida.

Tal vez se esté preguntando ahora en qué situaciones es necesario hacer trabajar el ordenador en base, por ejemplo, 19 ó 31. Esto puede darse en problemas de control de instrumentación en los que es a veces conveniente utilizar como base el número de instrumentos que hay que controlar: de ahí la necesidad de esa cantidad de bases numéricas, inexistentes en otros lenguajes. De todas formas no resulta muy racional trabajar en cuestiones de salida en bases extrañas. Puede ser por ello necesario poner en esos momentos el ordenador en base 10 sin olvidar la base de trabajo original. La secuencia que habrá que utilizar será:

BASE @	(reclama el valor en curso de BASE).
DECIMAL	(fuerza el sistema de numeración decimal).

...	(Se efectúan las operaciones).
...	(Para las que se necesita el sistema decimal).
BASE !	(sitúa de nuevo el valor de base numérica obtenido por la secuencia BASE@).

Puede ser aún más conveniente, para no perder la cuenta del lugar ocupado por la base numérica en el stack durante cálculos extensos, situar su valor en el RS, si se trabaja en el ámbito de una sola definición, o también definir dos nuevas palabras que utilicen dos variables definidas por el usuario. Por ejemplo, la variable

```
VARIABLE BASE<
```

y la palabra

```
: SAL BASE @ BASE< ! DECIMAL ;
```

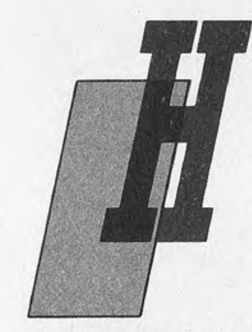
sirven para conservar el sistema numérico de base en curso y pasar a operar en decimal. El caso contrario

```
: ENTRA BASE< @ BASE ! ;
```

permite regresar al sistema de numeración originario.

CAPITULO XIV

CONCLUSION



emos llegado así al final de esta rápida visión del FORTH . Como habrán notado, en esta época de lenguajes estructurados con complejidades sintácticas y operacionales, el FORTH es el único que "impone" su uso desde un principio. Las palabras que hemos definido y compilado hasta ahora no son otra cosa que los procedimientos de otros lenguajes, que han sido aquí definidos de la manera más inmediata: con sólo nombrarlas e indicar a continuación lo que deben hacer.

Se ha dicho que el FORTH es un lenguaje de difícil lectura y compleja depuración. Esto es cierto a menos que se tomen algunas precauciones que deberían estar en la base de cualquier fase de programación. Vamos a dar entonces unos últimos consejos:

A) Incluya abundantes comentarios; no serán tenidos en cuenta en la fase de compilación, de manera que no ocuparán memoria ni harán más complejo el sistema operativo.

B) Defina palabras breves, mejor si son brevísimas y "cósalas" luego entre sí; depure entonces pequeñas secuencias. Esto es más racional y sistemático; además, una vez que las partes individuales son correctas resulta más fácil comprobar la validez del conjunto.

C) Fórmese una librería de base, con utilidades verificadas y de confianza. El FORTH, bajo este punto de vista, es menos rico que los demás en cuanto a "patrimonio" se refiere, pero esto ha sido pensado en parte para que el usuario pueda fabricarse palabras hechas a su medida y susceptibles de ser transformadas de una utilización a otra.

D) Trate de no traducir sus pensamientos, sino de "pensar en

FORTH"; verá que no resulta tan difícil como podría parecer, le bastará un poco de práctica.

E) Haga uso de las técnicas de recursividad. A pesar de que no hemos tratado de ellas en este libro, podrá encontrar bibliografía adecuada si decide empezar en serio con el estudio del FORTH. Para su información, la recursividad es la técnica en base a la cual un programa puede, en su ejecución, utilizarse a sí mismo para encontrar un resultado. Ya tratamos de ello en otro libro de esta colección dedicado a la programación estructurada.

Por último: en informática sea siempre sencillo, práctico y esencial. El FORTH en este aspecto es ciertamente un modelo ejemplar...

APENDICE A

FORTH 79 REQUIRED WORDS

En este apéndice está incluido el FORTH-79 REQUIRED WORDS SET obtenido del FORTH-79 Installation Normal. Se trata del conjunto mínimo necesario para cualquier sistema FORTH y destinado a asegurar la transportabilidad, que es el sueño de los lenguajes informáticos. El hecho de que hayamos puesto el estándar 79 y no el 83 se debe a que el primero está mucho más extendido y prácticamente disponible en cualquier sistema o procesador. Queremos recordar también que existe otro estándar, el FIG, que difiere de éste sobre todo en la nomenclatura de algunas palabras. Aquellas personas que quieran profundizar en los temas tratados podrán encontrar referencias útiles en la bibliografía que incluimos.

Hemos utilizado la siguiente norma para representar las modalidades ejecutivas de la palabra: en la secuencia

a b c - - - d

"a", "b", "c" representan el contenido del stack (en la que "c" es el TOS, es decir Top of Stack), - - - representa la ejecución de la palabra y "d" el contenido final del stack.

Manipulación de stack

DUP (n --- nn)	Duplica el número en el TOS
DROP (n ---)	Borra el número en el TOS

SWAP (n1 n2 --- n2 n1)	Intercambia los números en la cima del stack
OVER (n1 n2 --- n1 n2 n1)	Pone una copia del segundo número en el TOS
ROT (n1 n2 n3 --- n2 n3 n1)	Pone el tercer número en el TOS
PICK (n1 --- n2)	Copia el enésimo número en el TOS
ROLL (n --- n)	Pone el enésimo número en el TOS
?DUP (n --- n(n))	DUP sólo si el TOS es < > 0
>R (n ---)	Transfiere el TOS al Return Stack
R> (--- n)	Solicita un número del RS
R@ (--- n)	Copia el TRS en el Data Stack
DEPTH (--- n)	Cuenta los números del Stack

Comparaciones

< (n1 n2 --- flag)	Flag=1 (true) si n1 es menor que n2
= (n1 n2 --- flag)	Flag=1 si n1=n2
> (n1 n2 --- flag)	Flag=1 si n1>n2
0< (n --- flag)	Flag=1 si n es negativo
0= (n --- flag)	Flag=1 si n=0
0> (n --- flag)	Flag=1 si n es positivo
NOT (flag1 --- flag2)	Cambia el valor del flag. Equivalente a 0=
D< (d1 d2 --- flag)	Como [<] pero con números dobles
U< (u1 u2 --- flag)	Como [<] pero con números sin signo

Operaciones aritméticas y lógicas

+ (n1 n2 --- suma)	Suma algebraica
D+ (d1 d2 --- dsuma)	Suma algebraica de números dobles
- (n1 n2 --- resta)	Resta algebraica (n1-n2)
1+ (n --- n+1)	Añade 1 al TOS
1- (n --- n-1)	Resta 1 al TOS
2+ (n --- n+2)	Añade 2 al TOS
2- (n --- n-2)	Resta 2 al TOS
* (n1 n2 --- producto)	Producto
/ (n1 n2 --- cociente)	División (n1/n2). El cociente está truncado (no redondeado)
MOD (n1 n2 --- resto)	Deja en TOS el resto de n1/n2. El resto tiene el signo de n1
/MOD (n1 n2 --- resto cociente)	Deja el resto y el cociente en el TOS

*/ (n1 n2 n3 --- cociente)	Multiplica y luego divide (n1*n2/n3); el resultado está truncado. Los valores intermedios son de doble precisión
*/MOD (n1 n2 n3 --- resto cociente)	Igual que */ pero deja resto y resultado en el TOS
U* (un1 un2 --- uproducto)	Como * pero con números sin signo
U/MOD (ud un --- uresto ucociente)	Divide un número doble por uno sencillo, dejando un resto y cociente sencillos, ambos sin signo
MAX (n1 n2 --- max)	Deja el mayor de los dos números
MIN (n1 n2 --- min)	Deja el menor de los dos números
ABS (n --- (n))	Deja el valor absoluto
NEGATE (n --- -n)	Convierte el número en negativo (deja el complemento a 2)
DNEGATE (dn --- -dn)	Deja el complemento a 2 de un número doble
AND (n1 n2 --- AND)	AND lógico entre n1 y n2
OR (n1 n2 --- OR)	OR lógico entre n1 y n2
XOR (n1 n2 --- XOR)	OR exclusivo entre n1 y n2

Operaciones en memoria

@ (dir --- n)	Busca un número en la dirección
! (n dir ---)	Guarda un número en la dirección
C@ (dir --- byte)	Busca el LSB (least significant byte; Byte de orden bajo)
C! (n dir ---)	Idem pero lo guarda
? (dir ---)	Muestra el número de la dirección (equivale a @.)
+! (n dir ---)	Añade "n" al número presente en la dirección
MOVE (dir1 dir2 n ---)	Copia "n" números a partir de la dir1 en las posiciones a partir de la dir2, si n>0
CMOVE (dir1 dir2 n ---)	Idem pero con un byte
FILL (dir1 "n" byte ---)	Asigna a "n" bytes en memoria el valor del byte, partiendo de la dir1.

Estructuras de control

DO ... LOOP do:(valor final+1, v. de partida ---)	Estructura principal de bucle
I (--- índice)	Aloja el índice en curso del LOOP en el Data Stack
J (--- índice)	Igual con el bucle interior
DO ... +LOOP do:(límite, partida---)	Como DO-LOOP pero añade el valor del stack al índice
LEAVE (---)	Fuerza una salida prematura del bucle poniendo el límite igual al índice
IF ... (true) ... THEN if:(flag ---)	Si el TOS (flag)=true (verdad=1) se efectúa
IF ... (true) ... ELSE if:(flag ---)	Idem pero si el flag=0 efectúa la secuencia de ELSE
BEGIN ... UNTIL until:(flag ---)	Vuelve a BEGIN hasta que el flag se ponga verdad (=1)
BEGIN ... WHILE ... REPEAT while:(flag ---)	Efectúa el ciclo entre BEGIN y REPEAT hasta que el flag sea falso (=0); en tal caso el bucle, después del WHILE, salta a la instrucción después de REPEAT
EXIT (---)	Termina la ejecución de una definición; no debe ser utilizado fuera de un DO-LOOP
EXECUTE (dir ---)	Fuerza la ejecución de una dirección de diccionario en la dirección de compilación en stack

Operaciones de I/O

. (n ---)	Muestra el número del TOS
U. (nu ---)	Idem con número sin signo
CR (---)	Efectúa un salto a la línea siguiente (carriage return)
SPACE (---)	Deja un espacio (ASCII BLANK; 32)
SPACE (n ---)	Deja "n" espacios, si n>0
." (---)	Imprime un mensaje (cerrado por ")

TYPE (dir n ---)	Imprime una cadena de"n"caracteres a partir de la dirección
-TRAILING (dir1 --- dir2)	Elimina los espacios superfluos en la cadena presente en dir1
KEY (--- valor)	Lee del teclado un carácter y deja su valor ASCII en el TOS
EMIT (valor ---)	Imprime el carácter ASCII correspondiente al valor en el TOS
EXPECT (dir n ---)	Recibe una entrada por teclado de "n" caracteres (o que finaliza con CR (ASCII 13))
QUERY (---)	Recibe una entrada por teclado de una longitud de hasta 80 caracteres y lo sitúa en el buffer de entrada
WORD (valor --- dir)	Lee la palabra siguiente del flujo de entrada utilizando "valor" como delimitador

Operaciones de I/O en memoria de masa

LIST (n ---)	Lista la página "n" y asigna a "n" SCR
LOAD (n ---)	Carga (si no está ya presente) e interpreta la página "n"
BLOCK (n --- dir)	Deja la dirección de memoria del bloque n
BLK (--- dir)	Variable del sistema que contiene el número en curso de bloque
SCR (--- dir)	Variable de sistema que contiene el número de la página en curso
BUFFER (n --- dir)	Obtiene el buffer sucesivo de memoria y lo asigna al bloque "n"
UPDATE (---)	Advierte que el último buffer ha sido modificado
SAVE-BUFFER (---)	Salva todos los buffers modificados en la memoria de masa
EMPTY-BUFFER (---)	Pone todos los buffers disponibles

Palabras de definición

: nombre (---)	Inicia una definición
; (---)	Concluye una definición
VARIABLE nombre (---)	Crea una variable de 2 bytes llamándola nombre
nombre: (--- dir)	Devuelve la dirección de la variable

CONSTANT nombre (n ---) — Crea una constante con un nombre específico y almacena allí el valor de "n"

CREATE .. DOES> (does:(--- dir)) — Sirve para crear una nueva palabra de definición con ejecución inmediata

Otras palabras

CONTEXT (--- dir) — Variable del sistema que apunta al diccionario principal

CURRENT (--- dir) — Variable del sistema que apunta al diccionario de nuevas definiciones

FORTH (---) — Llama el diccionario principal y actualiza CONTEXT. En algunos sistemas con varios lenguajes hace entrar en el ambiente FORTH.

DEFINITION (---) — Asigna el diccionario CURRENT a CONTEXT

' (nombre) (--- dir) — Busca la dirección del nombre de la palabra

FIND (--- dir) — Deja la dirección de compilación de la palabra

APENDICE B

CODIGO ASCII

CARACTER	CODIGO BINARIO	CODIGO DECIMAL	CODIGO HEXADECIMAL	DEFINICION
NUL	0000 0000	0	00	Nulo
SOH	0000 0001	1	01	Principio de encabezamiento
STX	0000 0010	2	02	Comienzo de texto
ETX	0000 0011	3	03	Fin de texto
EOT	0000 0100	4	04	Fin de transmisión
ENQ	0000 0101	5	05	Pregunta
ACK	0000 0110	6	06	Acuse de recibo
BEL	0000 0111	7	07	Timbre (señal)
BS	0000 1000	8	08	Retroceso
HT	0000 1001	9	09	Tabulación horizontal
LF	0000 1010	10	0A	Cambio de renglón
VT	0000 1011	11	0B	Tabulación horizontal
FF	0000 1100	12	0C	Página siguiente
CR	0000 1101	13	0D	Retorno de carro
SO	0000 1110	14	0E	Fuera de código
SI	0000 1111	15	0F	En código
DLE	0001 0000	16	10	Encaje de transmisión
DC1	0001 0001	17	11	Mando de dispositivo auxiliar 1
DC2	0001 0010	18	12	Mando de dispositivo auxiliar 2
DC3	0001 0011	19	13	Mando de dispositivo auxiliar 3

CODIGO ASCII

CARACTER	CODIGO			DEFINICION
	BINARIO	DECIMAL	HEXADECIMAL	
DC4	0001 0100	20	14	Mando de dispositivo auxiliar 4
NAK	0001 0101	21	15	Acuse de recibo negativo
SYN	0001 0110	22	16	Sincronización
ETB	0001 0111	23	17	Fin de bloque de transmisión
CAN	0001 1000	24	18	Cancelación
EM	0001 1001	25	19	Fin de medio físico
SUB	0001 1010	26	1A	Sustitución
ESC	0001 1011	27	1B	Escape
FS	0001 1100	28	1C	Separador de fichero
GS	0001 1101	29	1D	Separador de grupo
RS	0001 1110	30	1E	Separador de registro
US	0001 1111	31	1F	Separador de unidad
	0010 0000	32	20	Espacio en blanco
!	0010 0001	33	21	Admiración
"	0010 0010	34	22	Comillas
#	0010 0011	35	23	Símbolo número (cancela)
$	0010 0100	36	24	Símbolo dólar
%	0010 0101	37	25	Porcentaje
&	0010 0110	38	26	"Ampersand"
'	0010 0111	39	27	Acento
(	0010 1000	40	28	Apertura de paréntesis
)	0010 1001	41	29	Cierre de paréntesis
*	0010 1010	42	2A	Asterisco
+	0010 1011	43	2B	Signo más
,	0010 1100	44	2C	Coma
–	0010 1101	45	2D	Guión (signo menos)
.	0010 1110	46	2E	Punto
/	0010 1111	47	2F	Símbolo división ("slash")
0	0011 0000	48	30	
1	0011 0001	49	31	
2	0011 0010	50	32	
3	0011 0011	51	33	
4	0011 0100	52	34	
5	0011 0101	53	35	
6	0011 0110	54	36	
7	0011 0111	55	37	
8	0011 1000	56	38	
9	0011 1001	57	39	
:	0011 1010	58	3A	Dos puntos
;	0011 1011	59	3B	Punto y coma
<	0011 1100	60	3C	Menor que
=	0011 1101	61	3D	Igual

CODIGO ASCII

CARACTER	CODIGO			DEFINICION
	BINARIO	DECIMAL	HEXADECIMAL	
>	0011 1110	62	3E	Mayor que
?	0011 1111	63	3F	Interrogante
@	0100 0000	64	40	"Atpersand", arroba
A	0100 0001	65	41	
B	0100 0010	66	42	
C	0100 0011	67	43	
D	0100 0100	68	44	
E	0100 0101	69	45	
F	0100 0110	70	46	
G	0100 0111	71	47	
H	0100 1000	72	48	
I	0100 1001	73	49	
J	0100 1010	74	4A	
K	0100 1011	75	4B	
L	0100 1100	76	4C	
M	0100 1101	77	4D	
N	0100 1110	78	4E	
O	0100 1111	79	4F	
P	0101 0000	80	50	
Q	0101 0001	81	51	
R	0101 0010	82	52	
S	0101 0011	83	53	
T	0101 0100	84	54	
U	0101 0101	85	55	
V	0101 0110	86	56	
W	0101 0111	87	57	
X	0101 1000	88	58	
Y	0101 1001	89	59	
Z	0101 1010	90	5A	
[	0101 1011	91	5B	Apertura de corchete
\	0101 1100	92	5C	Barra invertida ("Back slash")
]	0101 1101	93	5D	Cierre de corchete
^	0101 1110	94	5E	Acento Circunflejo
_	0101 1111	95	5F	Guión de subrayado
`	0110 0000	96	60	Acento inverso
a	0110 0001	97	61	
b	0110 0010	98	62	
c	0110 0011	99	63	
d	0110 0100	100	64	
e	0110 0101	101	65	
f	0110 0110	102	66	
g	0110 0111	103	67	
h	0110 1000	104	68	

CODIGO ASCII

CA-RAC-TER	CODIGO			DEFINICION
	BINARIO	DECIMAL	HEXADECIMAL	
i	0110 1001	105	69	
j	0110 1010	106	6A	
k	0110 1011	107	6B	
l	0110 1100	108	6C	
m	0110 1101	109	6D	
n	0110 1110	110	6E	
o	0110 1111	111	6F	
p	0111 0000	112	70	
q	0111 0001	113	71	
r	0111 0010	114	72	
s	0111 0011	115	73	
t	0111 0100	116	74	
u	0111 0101	117	75	
v	0111 0110	118	76	
w	0111 0111	119	77	
x	0111 1000	120	78	
y	0111 1001	121	79	
z	0111 1010	122	7A	
<	0111 1011	123	7B	Apertura de corchete
|	0111 1100	124	7C	Barra vertical
>	0111 1101	125	7D	Cierre de corchete
–	0111 1110	126	7E	Tilde
DEL	0111 1111	127	7F	Borrado, supresión

BIBLIOGRAFIA

STARDING FORTH.
L. Brodie. *Forth Inc-Pretince Hall.*

INTRODUCTION TO FORTH
K. Knecht. *Howard & Sams Co.*

FORTH PROGRAMMING
L. Scanlon. *Howard & Sams Co.*

AIM-65 FORTH USER'S MANUAL
Rockwell.

DISCOVER FORTH
T. Hogan. *MacGraw-Hill.*

Nota: Algunos de estos libros son importados también por Díaz de Santos.

BIBLIOTECA BASICA INFORMATICA

INDICE GENERAL

10 **Sistemas operativos y software de base**
Qué son, para qué sirven. Unos desconocidos muy importantes.
11 **Sistema operativo CP/M**
Uno de los sistemas operativos para microprocesadores de 8 bits de mayor difusión en el mercado.
12 **MS-DOS: el estándar de IBM**
Sistema operativo para el microprocesador de 16 bits 8088, adoptado por el IBM-PC.
13 **Paquetes de aplicaciones. Software "pret a porter"**
Características y peculiaridades de los más importantes paquetes de aplicaciones.
14 **VisiCalc: una buena hoja de cálculo**
Interioridades y manejo de una de las hojas de cálculo más usadas.
15 **Dibujar con el ordenador**
Profundizando en una de las facetas útiles y divertidas que nos ofrecen los ordenadores.
16 **Tratamiento de textos... para escribir con el ordenador**
Cómo convertir su ordenador en una máquina de escribir con memoria y todo tipo de posibilidades.
17 **Diseño de juegos**
Particularidades características de esta aplicación de los ordenadores.
18 **LOGO: la tortuga inteligente**
Un lenguaje conocido por su «cursor gráfico», la tortuga, y sus aplicaciones pedagógicas al alcance de su mano.
19 **Paquetes integrados: Lotus 1-2-3 y Simphony**
Estudio de dos de los paquetes integrados (Hoja de cálculo+base de datos+...) más conocidos.
20 **dBASE II y dBASE III**
Cómo aprovechar las dos versiones más recientes de esta importante base de datos.
21 **Bancos de datos (I)**
Peculiaridades de una de las aplicaciones de los ordenadores más interesantes y que más dinero mueven.
22 **Bancos de datos (II)**
Profundizando en sus características.
23 **FORTH: anatomía de un lenguaje inteligente**
Principales características de un lenguaje moderno, flexible y de amplio uso, en la robótica.
24 **BASIC y tratamiento de imágenes**
Todo lo que en **¡Y llegó el BASIC!** no se pudo ver sobre las imágenes y gráficos en el BASIC.
25 **Los ordenadores uno a uno**
Un amplio y completo estudio comparativo.
26 **Cálculo numérico en BASIC**
Una aplicación especializada a su disposición.
27 **Multiplan**
Cómo hacer uso de este moderno paquete de aplicaciones.
28 **FORTRAN y COBOL**
Dos lenguajes muy especializados y distintos.
29 **Softest.** Los programas a examen
30 **Cómo realizar nuestro propio banco de datos**
Conocimientos necesarios para poder fabricar un banco de datos a nuestro gusto y medida.

NOTA: Ingelek, S. A. se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de la colección.

NOTAS

Hay quienes al acercarse por primera vez al FORTH se sienten extrañados de sus peculiaridades y un poco perdidos, llegando incluso a tacharlo de «raro» y abandonar cualquier intento de profundizar en él. Sin embargo, todos aquellos que sean capaces de superar esta primera barrera de «lo distinto» se verán finalmente compensados.

El FORTH es un lenguaje moderno, con el cual es natural la programación «de abajo hacia arriba», claro (una vez comprendida su filosofía), que permite obtener el máximo provecho del ordenador y aprovechar, desde un lenguaje de alto nivel, la rapidez y potencia del lenguaje máquina. En este libro le explicaremos el estilo de programación característico del FORTH, su filosofía y lo que podrá o no hacer con él.

395 pts.
(incluido IVA)